マイクロコンピュータの基本ソフトウェア

実習CP/M®

村瀬康治——著

アスキー出版局

実習CP/M®

村瀬 康治　著

アスキー出版局

*CP/M Learning System*全3巻の構成

この「CP／M Learning System」全３巻は，次のように構成されています．

入門CP／M………CP／Mがどの様なものであるかを解説し，CP／Mを使うための基礎知識，日常よく使う各コマンドの実習などをやさしく具体的に解説します．
今までCP／Mについては全く知らなかった読者でも，本書によりCP／Mの概要を理解し，一通りCP／Mが使えるようになれるよう配慮されています．

実習CP／M………CP／Mの全コマンドと，そのほとんどすべての使い方を徹底的かつやさしく，具体的に実習しながら解説します．また，CP／Mのハードウェアおよびソフトウェアの構成についても解説し，CP／Mの実用例として，CP／Mアセンブラによるマシン語開発の全過程を実習します．
本書は，読者が本格的にCP／Mを使うようになった場合，CP／Mのコマンド・ハンドブックとして，随時参照することになるでしょう．

応用CP／M………CP／Mをさらに深く広く応用する場合についての解説書です．マクロ・アセンブラによるマシン語開発，システム・コール，各種高級言語の使用例，アプリケーションやユーティリティの実行なども，分かり易い実行例に基づいて解説します．さらにCP／Mの内部構造，BIOSの詳細など，本格的なCP／Mユーザーとして，いずれは必要となる知識を提供します．

各巻はそれぞれにまとまっていますので，必要に応じてどの巻を手にされても利用することができます．

本シリーズは，CP／M version ２．２を基にして書かれていますが，旧バージョンである，version １．４を使う場合のことを考慮し，共通でないコマンドについては，そのつど注意書きを付け加えてあります．

本書「実習CP/M」は

本書は，本書のみで，CP／M の実用書として活用できるように，すべてのコマンドの，そのほとんどの使い方を実例で具体的に解説しています．よって，既刊「入門 CP／M」と重複する部分も敢えて解説しています（より高度に解説）．

しかし，基本的には「入門 CP／M」の意図する，CP／M の基礎知識は，すでにあるものとして構成されていますので，〝これから CP／M を……〟という方は，ぜひ「入門 CP／M」も合わせてご覧下さい．

本書は，具体的な実行例リストを，紙面の許す限り，たくさん載せました．これらのリストは，CP／Mユーザーによるキー入力部分と，それに対するCP/Mの応答を，よく対比して読んで下さい．きっと，文章による解説以上の理解が得られることと思います．

また，本書で示されている実行例は，それぞれのコマンドを解説するための，一つの例にすぎず，使い方によって，いくらでもバリエーションが可能です．本書は，そのHow To Useの基本を，わかりやすく提供することを第1の目的としています．

著者まえがき

既刊の CP／M シリーズ第1巻「入門 CP／M」は，各方面の多くの方々から，大変好評を頂いており，「早く続巻を出してほしい」という，たくさんのハガキがアスキーに寄せられました．筆者にとって，こんなにうれしいことはありません．なのに，本書「実習 CP／M」の発刊が少し遅れましたことをお詫び致します．

本書は，世の中の CP／M の実用参考書として，中心的役割を果す充実したものにまとめられています．すべてを実例によって，これほど多くの使い方を分かり易く具体的に解説したものは，今まで内外ともにありません．今までのマニュアルや参考書では，触れられていなかったり，ウヤムヤであった高度なコマンドの使い方なども，本書により，実例として読者の前に明らかになることでしょう．

今，パーソナル・コンピュータにおいて，日米間のハードウェア・ギャップは完全に埋り，例えば，NEC の PC-8800 や OKI の if 800 Model 30 などを始めとする，日本の第2期のパーソナル・コンピュータとでも言うべき各社の製品は，むしろ先を行っていると言えるでしょう．

しかし，アメリカで次々と生まれる各分野の優れたソフト(最近は，ソーシム社の「スーパーカルク」，マイクロソフト社の「マルチプラン」，マイクロプロ社の「カルクスター」など，ビジネス・ソフトの開発が目につく．いずれも CP／M 上で実行され，今までの内外の同種のソフトに比べ格段に多くの機能がある．)に接し，肌で感じることは，日米間のソフトウェア・ギャップは，〝むしろ広がって行くのではないか〟という危惧の念です．

筆者は，パーソナル・コンピュータにおけるこのソフトウェア・ギャップを埋める鍵は，CP／M の普及を置いて他にはないと思っています．CP／M (あるいは CP／M 相当以上の OS)を使いこなせなければ，コンピュータに対する良いセンスを養うことはできないでしょう．一般の DISK BASIC などの，必要最少限で付け足し程度の OS しか持たないものでは，コンピュータを活かした経験はできません．何のためのディスク・システムなのか，もったいない限りです．

現在のアメリカのソフトウェア水準(パーソナル・コンピュータの)は，一般の多くの CP／M ユーザーが支えていると，明言することができます．ごく低価格の学習用マシンを除き，コンピュータを仕事に使っている人は，ほとんどが CP／M をベースにしています．要するに彼等は，コンピュータの基本(まともな OS の意味)の上に立って，BASIC 言語なり，アプリケーションなりを実行しているのです．日本のメーカーが，CP／M の走らない BASIC マシン(ディスク付)を輸出しようとしても，相手にされないと言うのは当然の話なのです．

ソフトウェア・ギャップ……，IC1 個，抵抗1本の増減を問題にする OEM のエンジニア諸兄は別

として，一般のユーザーが，コンピュータの機種選びに，CPU チップの優劣などを第1の問題にする時代ではありません．もっともっとソフトウェア環境(OS とその上で走るソフトウェア)に目を向けなければならないのではないでしょうか．「日本人は只の金物職人」と思われない為にも…….

しかし，本書，CP/M シリーズに対する反響からも，予想以上に多くの方々が，CP/M (つまりはソフトウェア)に大きな関心を持っていることを知り，この差も近い将来，急加速で追いつくことができると確信しています．

1982年の末までに，日米間の CP/M マシン台数は50万台を軽く突破すると予測されています．本書が，これら多くの CP/M ユーザーから，メインの実用書として愛用されることを願っています．

堅い本なので，〝少しでも彩りを〟と思い，挿絵を斎藤智子さんに描いていただきました．武蔵野の風景とのことです．

1982年1月　　村瀬康治

コマンド・インデックス

目次

★　5章　CP/Mによるマシン語開発実習――183

実習一覧

PIP（周辺装置間のデータ転送プログラム）

1章　CP/Mのハードウェア構成

1.1 最も基本的なハードウェア構成

CP／M を走らせるために必要な最小限のハードウェア構成を次に示します.

(1) 8080, 8085, Ｚ80系の CPU を持つコンピュータ本体.

CP／M に, それぞれの CPU に対するバージョンがあるわけではなく, 8080用に書かれたものですが, 8085, Ｚ80は8080のマシン語に対して, アッパー・コンパチブルですから, そのまま使用できます. 6502, 6809など, 他の CPU には, 〝Ｚ80カード〟などのハードウェア・オプションが必要となります.

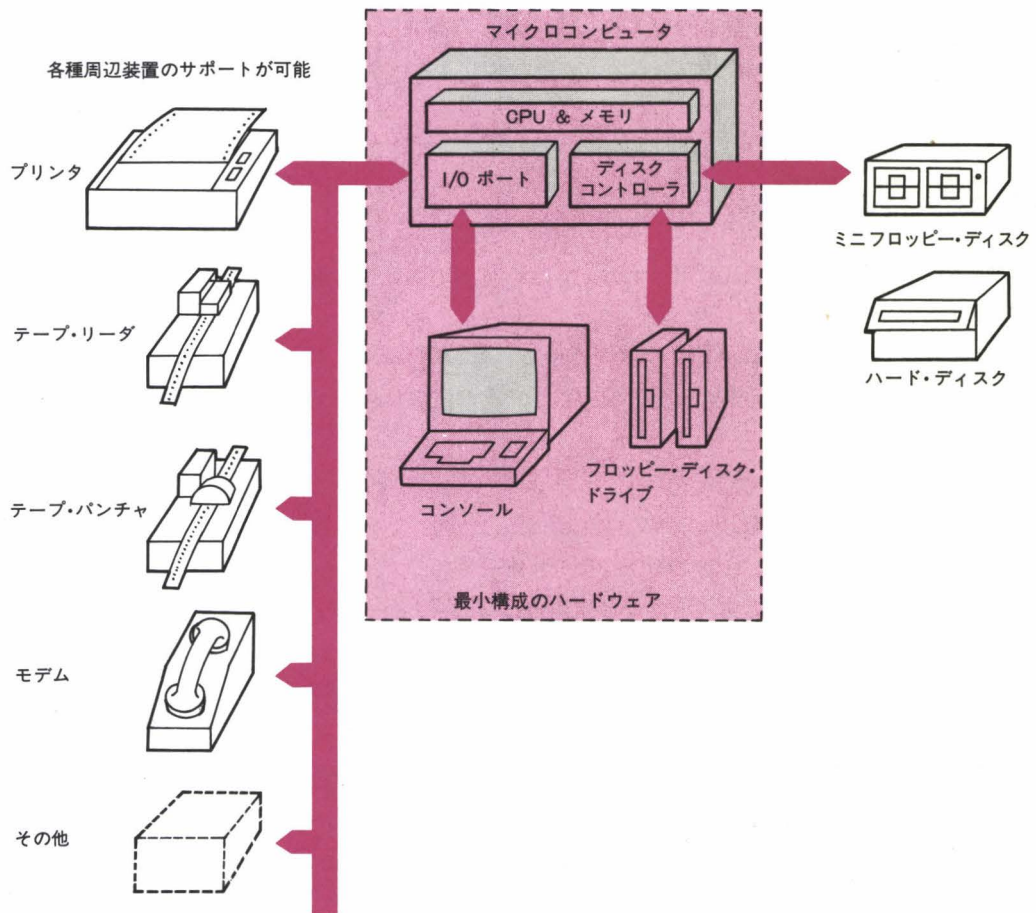



Figure-1.1.1 CP／Mマシンのハードウェア構成と周辺装置の拡張

RAM はアドレス 0 Hから連続していなければならず，最小で20Kバイト，実務には48Kバイト程度は必要です．

(2) コンソール（キーボード付きの CRT ターミナル）．または，それの代用となるもの（VRAM など）．

(3) フロッピー・ディスク・ドライブ（ディスク・コントローラを含む）．
標準ドライブとしては 8 インチ片面単密度ですが，両面倍密度やミニフロッピー・ディスク，あるいはハード・ディスクなど，いずれでも使用できます．ドライブの数は 1 台でも CP／M を走らせることはできますが，ディスク・システム（フロッピー・ディスクの場合）は，CP／M に限らず汎用機としての実務には 2 台以上でないと能率が上らず，仕事にならないでしょう．

以上挙げた(1) (2) (3)が最も基本的な CP／M マシンの構成であり，Figure-1.1.1 の点線内がその最小構成を示しています．この構成でも，コンピュータのメモリ容量さえ十分であれば，CP／M のもとで走るほとんどのソフトウェアが実行可能です．

1.2 周辺装置の拡張

CP／M マシンは，一般の BASIC マシンでは不可能であった多種多様な周辺装置を，簡単な操作で取り扱うことができます．

周辺装置の扱いについては，「PIP コマンド」と「STAT コマンド」に深く関係しますので，詳細はそれぞれのコマンドの項で解説しますが，ここでその概要について述べておきましょう．

ディスク・ドライブ

ディスク・ドライブの種類は，ミニフロッピー・ディスクから大容量のハード・ディスクまで，片面，両面，密度など，どのような規格のものでも使用可能です．接続可能なドライブ数は16台（CP／M Version1.4 は 4 台）までです．1 ドライブ当り，CP／M がコントロール可能な最大容量は 8 Mバイトですが（CP／M Version2.0 以上），フィジカルなドライブ数を減らして 1 台のドライブの容量を増すことは可能であり，例えば，ハード・ディスクを 1 台のみ接続する場合には，最大128Mバイトの容量のものまでコントロール可能になります（1 つのディスクを，A:，B:，C:，……に分割して使用する．16台×8 M＝128M)．但し，1 つのファイル長はやはり 8 Mバイト以下でなくてはなりません．

雑誌などの CP／M の記事に，時々「CP／M はファイルの容量不足」ということが書かれることがありますが，何と比較しているのでしょうか．CP／M は Version2.0 (1979年発売，現在は2.2である.)から，1 ファイル最大 8 Mバイトの容量を持つことができるのです．但し，ファイルは別のドライブにまたがって作ることはできません．フロッピー・ディスクの容量の小さいものを使用している

場合は，そのドライブの最大容量が1ファイルの最大容量となりますので，制約を受ける場合もあるでしょう（CP／M の次の Version からは，1ファイル64Mバイトに拡張され，異なるドライブ上のファイルとのチェインが可能となる）.

ディスク・ドライブを複数台接続する場合，同機種のものでなくてもかまいません．例えばミニと8インチ・フロッピー・ディスク，それにハード・ディスクの3種類を Figure-1.2.1 のように使うこともできます．実際に，4台のミニフロッピーと4台の8インチ・フロッピーを同時にコントロールできるように作られているパーソナル・コンピュータもあります．

各種のディスク・ドライブが同時に接続可能です.

例えば，このようなドライブ名の割り付けをすることも，BIOS の書き方により自由にできる.

A： B： C： D： E：

パーソナル・コンピュータ本体に組み込みのミニフロッピー・ディスク

増設用の8インチ・フロッピー・ディスク

特別に増設した大容量のハード・ディスク

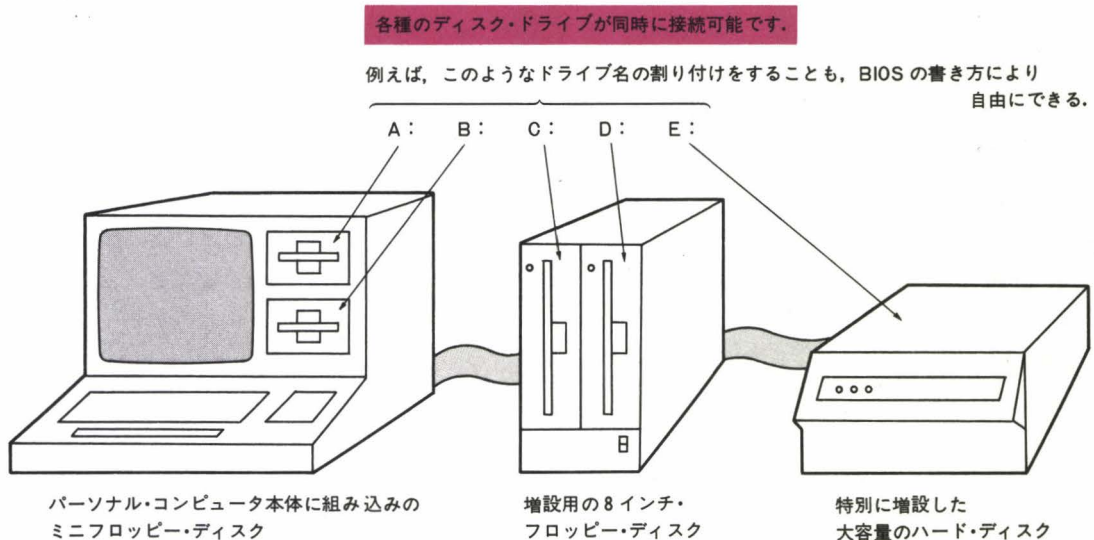



Figure-1.2.1　各種ディスク・ドライブを同時に接続

ディスク以外の周辺装置 <ロジカル・デバイス，フィジカル・デバイス>

周辺装置には大きく分類して4種の〝ロジカル・デバイス〟があり，そのそれぞれに対して，さらに4つの〝フィジカル・デバイス〟を任意に割り当てることができます．つまり計16種の周辺装置を，常時接続して取り扱うことが可能なのです．ロジカル・デバイスの4つのグループの，それぞれの意味を次に示します．

- 〝CON:〟というロジカル・デバイス名で代表される，コンソールに類する4種類のコンピュータとの対話用入出力装置．（TTY:（テレタイプ） CRT:（スクリーン） BAT:（バッチ・プロセス） UC1:（ユーザー・コンソール＃1））
- 〝RDR:〟というロジカル・デバイス名で代表される，紙テープ・リーダなどに類する4種類の入力装置．（TTY:（テレタイプ） PTR:（ペーパーテープ・リーダ） UR1:（ユーザー・リーダ＃1） UR2:（ユーザー・リーダ＃2））

● "PUN:" というロジカル・デバイス名で代表される，紙テープ・パンチャなどに類する4種類の出力装置．　(TTY:(テレタイプ)　PTP:(ペーパーテープ・パンチャ)　UP1:(ユーザー・パンチャ#1)　UP2:(ユーザー・パンチャ#2)　)
● "LST:" というロジカル・デバイス名で代表される，プリンタなどのリスト装置に類する4種類の出力装置．　(TTY:(テレタイプ)　CRT:(スクリーン)　LPT:(ラインプリンタ)　UL1:(ユーザー・リスト・デバイス)　)

以上の4×4＝16種類の周辺装置を取り扱うことができます．(　)内のフィジカル・デバイスは，それぞれ，TTY とか PTP とか LPT などの記号名で呼ばれますが，それらの実際の装置が，テレタイプであり，ペーパーテープ・パンチャであり，ライン・プリンタである必要はありません．例えば記号名が"TTY"であっても，ハードウェア，ソフトウェア共に接続が可能であれば，高速の CRT ターミナルを接続してもよいし，電話線を使って交信する，モデムを接続してもよいのです．

モデムを使用する場合は "RDR:" の1つに，その受信部を設定し，"PUN:" の1つにその送信部を設定する場合もあるでしょう．これらフィジカル・デバイスのポート形式は，セントロニクス規格，RS-232C，IEEE-488など，ユーザーの要望に合わせて自由であり，それらのI／Oルーチンを BIOS に設けておけば，どんな装置でもCP／M マシンの周辺装置として取り扱うことが可能になります．

このように CP／M マシンは，周辺装置とのインターフェイスさえ設ければ，完備されたデータ転送用のコマンド（PIP）を使って，多くの周辺装置とのデータ転送が自由に行えるのです．

周辺装置の割り付けに関しては，第4章の「STAT コマンド」の項を，データ転送に関しては「PIP コマンド」の項を必読下さい．

CP/Mが直接に呼ぶことができるのは，CON：，RDR：，PUN：，LST：の4つのロジカル・デバイスであり，ユーザーは事前に，STATコマンド，あるいはI/Oバイト操作などで，各ロジカル・デバイスに対して，それぞれのフィジカル・デバイスを指定しておいてから，目的の装置とのデータ転送を行う．但しPIPコマンドではフィジカル・デバイスも直接呼べる．
フィジカル・デバイス名は特に意味はなく，"TTY" であっても，実際にTTYである必要はなく，BIOSの書き方次第で，ユーザーの必要とするポート形式を自由に当てはめればよい．パーソナル・コンピュータ用のCP/Mは，すでに何種類かの入出力ポートがサポートされているものが多い．

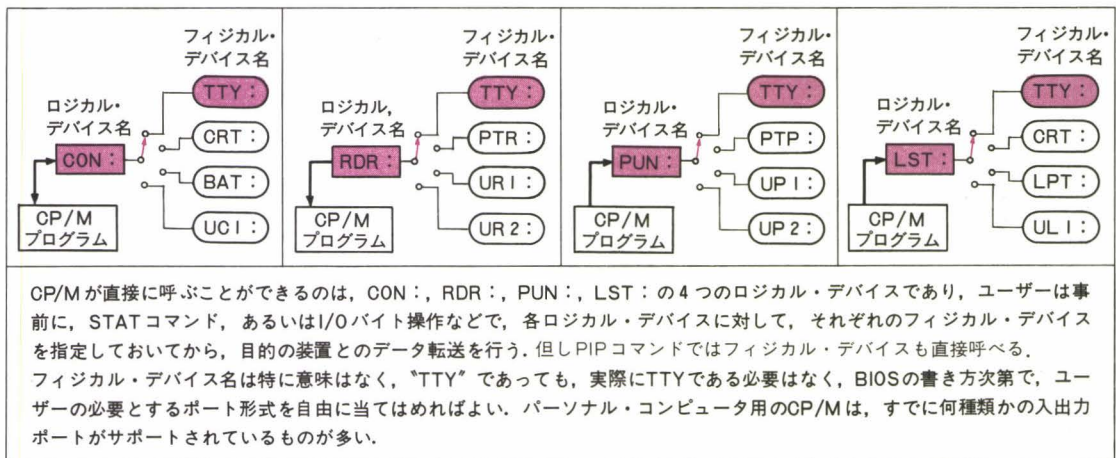



Figure-1.2.2　ロジカル・デバイスに対するフィジカル・デバイスのアサイン(割り当て)

IO バイトについて

アドレス03Hの1バイトは〝IO バイト〞と呼ばれ，前述の Figure-1.2.2 に図示されている，各ロジカル・デバイスに対するフィジカル・デバイスの割り付けを，通常用いる STAT コマンドを使用せずに，直接この〝IO バイト〞を操作することにより行うことができます．実は，STAT コマンドによるフィジカル・デバイスの割り付け（第4章「STAT コマンド」の項，「実習⑨」を必読）とは，STAT コマンドを介して，この IO バイトを操作することに他ならないのです．

STAT コマンドにより，IO バイトの値が変化する実例を次に示しましょう．

```
A>STAT DEV: ↲ ------イニシャル時のアサイン状況を見る.
CON: is TTY: ┐
RDR: is TTY: │---- 各CP/Mマシンによって異なるが，例えば
PUN: is TTY: │     このように割り付けられていたとする.
LST: is TTY: ┘

A>DDT↲ ------DDTを起動.
DDT VERS 2.2
-D3,3↲ ------アドレス03Hを見る.
0003 00 . --- "00" であった．Figure-1.2.4表を参照.
-^C --------- DDTを終る.

A>STAT RDR:=PTR: ↲ ----RDRにPTR:をアサインする.

A>STAT PUN:=PTP: ↲ ----PUNにPTP:をアサインする.

A>STAT DEV: ↲ -------- 現在のアサイン状況を見る.
CON: is TTY:
RDR: is PTR: ┐------ この2つが先ほどアサインされたもの.
PUN: is PTP: ┘
LST: is TTY:
A>

A>DDT↲ ------DDTを起動.
DDT VERS 2.2
-D3,3↲ ----- アドレス03Hの変化を見る.
0003 14 . ---"00"→"14" に変化している．Figure-1.2.4表を参照.
-^C --------DDTを終る.
A>
```

Figure-1.2.3 STAT コマンドによるフィジカル・デバイスの割り付けで，変化する IO バイトの例．

上記の操作で，IO バイトが00→14と変化しました．この意味は次の表により明らかになるでしょう．

	LST:	PUN:	RDR:	CON:	ロジカル・デバイス名
	bit 7 ¦ bit 6	bit 5 ¦ bit 4	bit 3 ¦ bit 2	bit 1 ¦ bit 0	IOバイト(アドレス03H)のbit ポジション
[バイナリ・データ] フィジカル・デバイス名	[0 0] TTY:	[0 0] TTY:	[0 0] TTY:	[0 0] TTY	
	[0 1] CRT:	[0 1] PTP:	[0 1] PTR:	[0 1] CRT:	
	[1 0] LPT:	[1 0] UP1:	[1 0] UR1:	[1 0] BAT:	
	[1 1] UL1:	[1 1] UP2:	[1 1] UR2:	[1 1] UC1:	
Figure−1.2.3の例で, IOバイト=14Hになる理由	0 0 0 1 (TTY:) 1 (PTP:)		0 1 0 0 (PTR:) 4 (TTY:)		

Figure-1.2.4 IOバイトにおけるロジカル・デバイスとフィジカル・デバイスの表現

このIOバイトの操作により，各種周辺装置の選択はSTATコマンドに依らず，ユーザー・プログラムの中で自由に行うことができます．例えば，「UR1：とUR2：の両RDR：入力ポートを監視しながら，入力のあった方からデータを受けとり，ディスクにファイルし，ファイル終了後その内容を今度は，PTP：とUP1：の両出力ポートに順に送り出す.」というような仕事も，アドレス03Hのデータを任意に変えることにより，ユーザー・プログラムで自動的に行わせることができるのです．

但し，IOバイトの値による，デバイスの選択に関してのソフトウェア（例えば，IOバイトが14Hであった場合，Figure-1.2.4の下段にような選択を行うためのルーチン，Figure-1.2.2に示すそれぞれの〝スイッチ〟の部分にあたるもの.）は，BIOSに組み込まれていなければなりません．標準CP／Mでは，4つのロジカル・デバイスのルーチンが組み込まれているだけで，そこまでは作られていません．BIOSを設計される方が，独自に作る必要があります．しかし，パーソナル・コンピュータで各種のIOポートを内蔵しているマシンのCP／Mは，たいていこのIOバイトがすでにサポートされていますので，そのままでIOバイトを，フィジカル・デバイスの選択に利用することが可能です（各マシンのCP／Mマニュアルをご覧下さい)．

2章　CP/Mのソフトウェア構成

本章は CP／M の内部について，ある程度理解していただくために，CP／M の構造，メモリマップ，ディスケット上の CP／M システム，CP／M 起動のメカニズムなどについて，その概要を解説します．

本章の内容は，CP／M 上で各種高級言語を使ったり，ワード・プロセッサやビジネス・プログラムなどのアプリケーション・プログラムを実行するだけであれば，さほど必要な知識ではありません．しかし，アセンブラで書かれたソフトウェアを開発したり，CP／M の BIOS を変更して，新しい CP／M システムを作成しようとする場合などは，本章での知識が必要になってきます．

2.1 CP/Mシステムの構成

通常〝CP／M システム〟と呼ぶ場合，〝システム〟とはハードウェアの〝装置〟を指すのではなく，CP／M の〝OS〟（オペレーティング・システム）のことを言います．

CP／M システムは次の 4 つのモジュールから構成されています．

BIOS（バイオス） **(Basic I/O System)**

マシンのハードウェアと，CP/Mとの入出力に関するインターフェイス部．

BDOS（ビードス） **(Basic Disk Operating System)**

ディスクのファイル管理などを行うCP/Mの心臓部．

CCP（シーシービー） **(Console Command Processor)**

キーボードからのコマンド入力を解釈実行する部分．ビルトイン・コマンドはこの中に含まれている．

TPA（ティーピーエー） **(Transient Program Area)**

トランジェント・プログラムやユーザー・プログラムがロードされるエリア．

以上の 4 つのモジュールがメモリ上の所定の位置に配置され，CP／M のすべての働きがコントロールされているわけです．では具体的にどのようなメモリマップになっていて，各部はどのような働きをするのでしょう．その様子を Figure-2.1.1 に示します．

Figure-2.1.1 の各部の実アドレス値は，58K　CP／M の場合の値ですが，これはそれぞれのコンピュータの使用可能メモリ容量により，この図で示されている CC00H ～ E7FFH 間のシステム部が前後にリロケートされ，通常は，許される最高位アドレスに設定されます（リロケートについては，第 4 章の「MOVCPM コマンド」を参照）．

CP／M サイズの大小による違いは，TPA と呼ばれるユーザー・プログラム・エリアが，広いか狭いかだけの問題であり，CP／M の基本動作には何の違いもありません．しかし，ユーザー・エリアは広い方が良く，例えば White Smith 社の C コンパイラのように大きなプログラムになると，〝59K バ

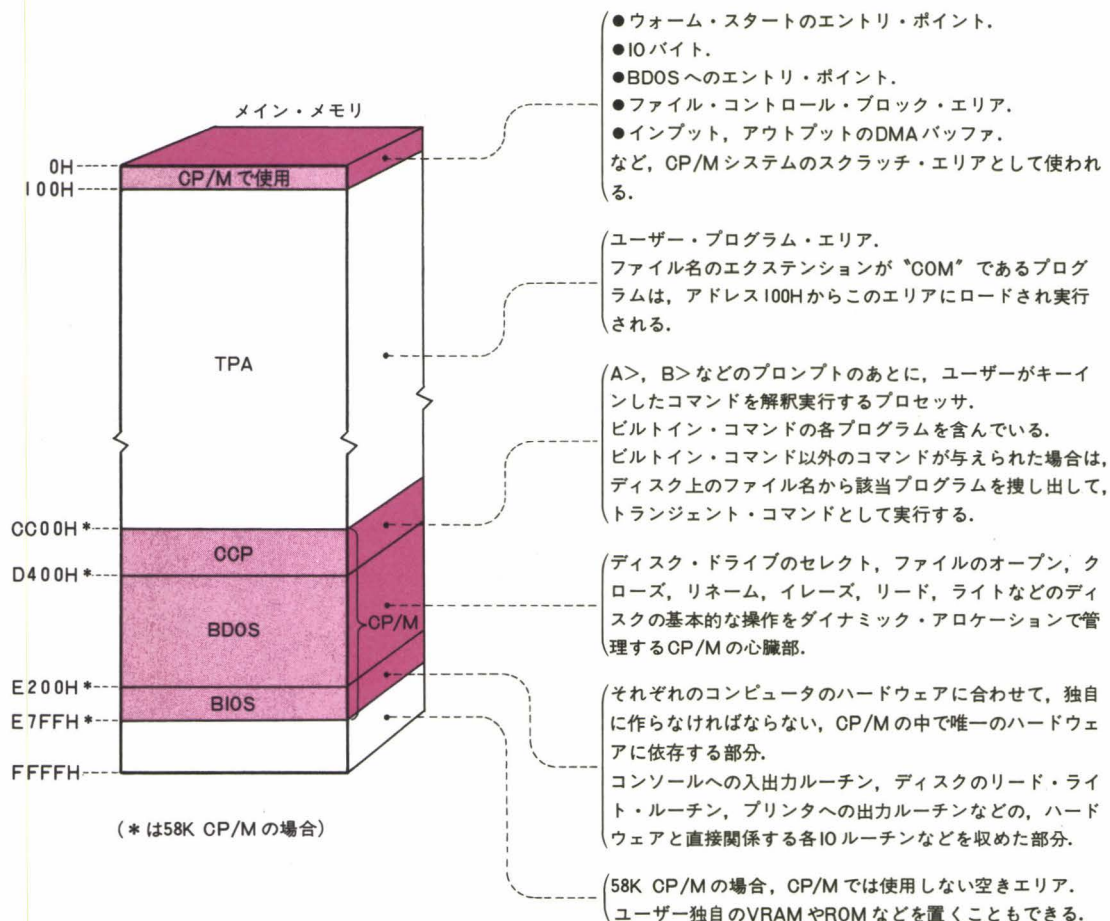



Figure-2.1.1　CP／Mシステムの構成

イト CP／M 以上で使うように″ などと指示があったりします．プログラムによっては注意が必要です．基本的には，CP／M サイズは最小20Kバイト（BIOS の終りが 4FFFH）から，最大64Kバイトまで任意の大きさに設定することができます．

これらの CP／M サイズの関係を図示したものを Figure-2.1.2 に示します．

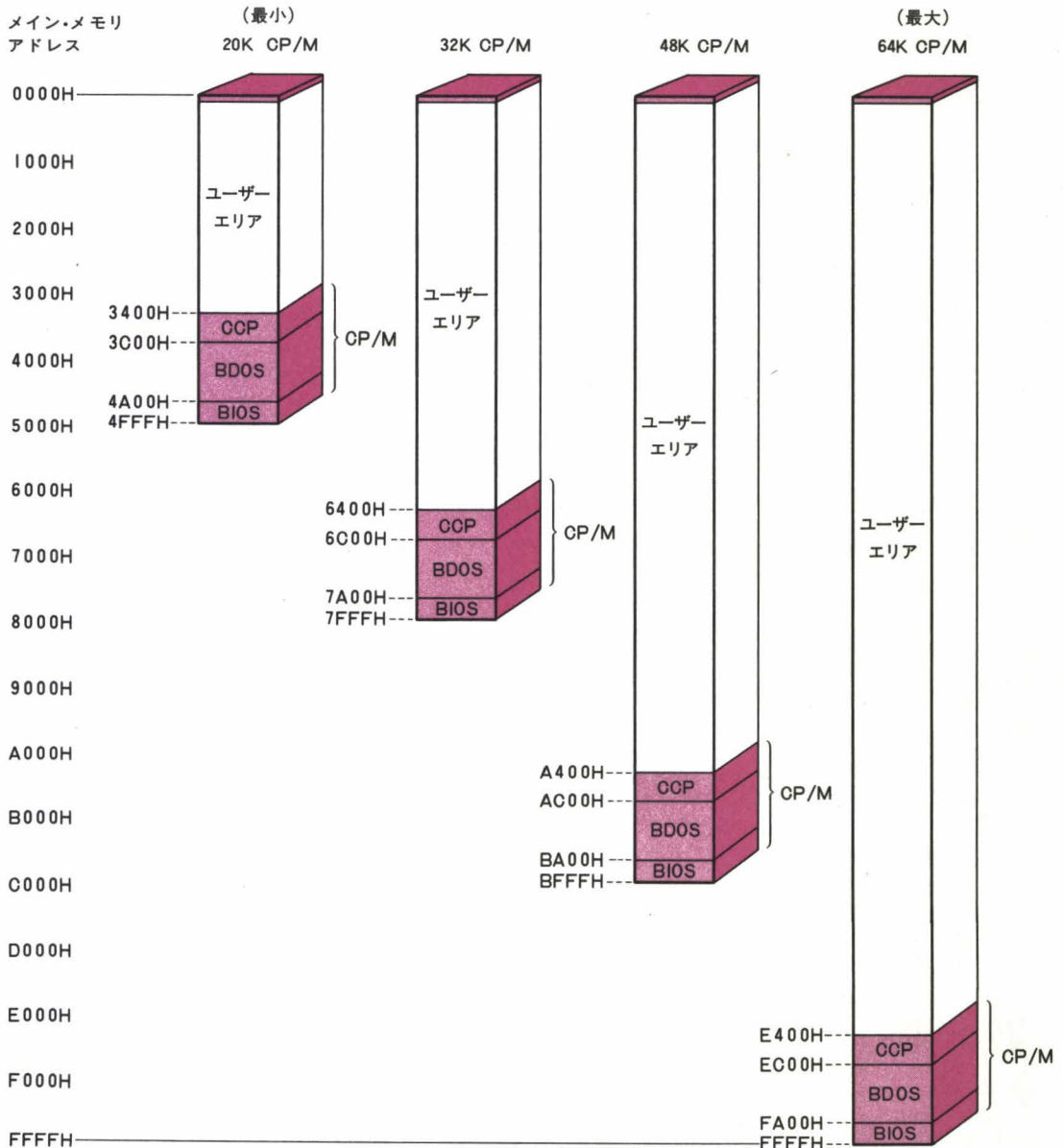



Figure-2.1.2 CP/Mのメモリ・サイズによる比較

2.2 CP/Mの動作の流れ〈ビルトイン・コマンドとトランジェント・コマンド〉

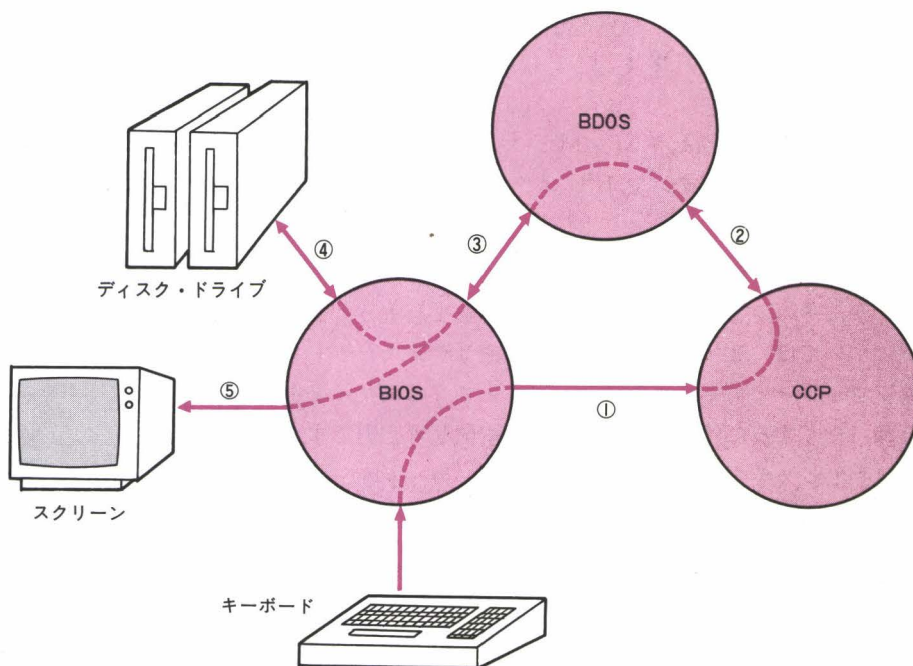
CP／Mが何らかのコマンドを実行する場合，その内部の流れはどうなっているのでしょう．簡単な例として，ビルトイン・コマンドである“TYPE”コマンドを例に解説しましょう．

TYPEマンドは，6つのビルトイン・コマンドの内の1つで，ディスク上の任意のアスキー・ファイルをコンソールにタイプアウトします．

ここで，DUMPプログラムのアセンブリ・ソース・ファイルである“DUMP．ASM”をタイプアウトする実行例をFigure-2.2.1に示します．下線部をキーインし“↲”でリターンすると，以下のリストがコンソールに表示されます．実行が終ると，自動的にCP／Mにもどり，再び“A＞”のプロンプトが出力されています．

```
A>TYPE DUMP.ASM ↲

;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
    .
    .
    .
;       FIXED MESSAGE AREA
SIGNON: DB      'FILE DUMP VERSION 1.4$'
OPNMSG: DB      CR,LF,'NO INPUT FILE PRESENT ON DISK$'

;       VARIABLE AREA
IBP:    DS      2       ;INPUT BUFFER POINTER
OLDSP:  DS      2       ;ENTRY SP VALUE FROM CCP
;
;       STACK AREA
        DS      64      ;RESERVE 32 LEVEL STACK
STKTOP:
;
        END

A>
```

Figure-2.2.1 TYPEコマンドでタイプアウトされたDUMPプログラムのソース・ファイル

さて,このTYPEコマンドが実行される場合の「キーボードからコマンドの入力⇒ディスク上のファイル〝DUMP.ASM〞の内容の読み出し⇒スクリーンへの表示」CP/M内部の流れを，BIOS, CCP, BDOSの3つのモジュールに関して図示解説したものをFigure-2.2.2に示します.

① キーボードからのコマンド・ライン〝TYPE DUMP.ASM ⏎〞がBIOSを通してCCPに送られる.

② CCPは送られてきたコマンド・ラインを解読し，制御をCCP内組み込みの〝TYPE〞プログラムに渡す.
TYPEプログラムはディスク上のファイル〝DUMP.ASM〞を読み出すために，ディスクの操作をBDOSに依頼する.

③④ BDOSはディスク・ドライブのハードウェアに直結したBIOS部を通してディスクをアクセスし，ファイル内容をメモリに読み込む.

⑤ メモリに読み込まれたファイルの内容は，BIOSを通して逐次スクリーンに出力される.

注） 上記は基本的な流れを示したもので，補助的な詳細は略してあります.

Figure-2.2.2 ビルトイン・コマンド(〝TYPE DUMP.ASM〞)が実行される場合のCP/M内部の流れ

Figure-2.2.2の例はCCPモジュール内に常駐しているビルトイン・コマンドを取り上げていますが，トランジェント・コマンドや他のアプリケーション・プログラムなどの場合には，CCPはまず，実行しようとするプログラムをディスク上から捜し出し，TPAにロードしなければなりません.目的のプログラムが見つかって,100Hからのメモリへのロードが終り,制御をそのプログラムに渡せば(ロ

ードし終った後，CCP は直ちに 100H にジャンプする） CCPの役目は終ります．もし長いプログラムであれば，CCP のエリアに侵入し，そこを破壊して使ってもいいのです．例えばトランジェント・コマンドの DDT は，最終的には CCP のエリアにオーバー・ライトされるのです．第 4 章の「DDT コマンド」を参照．

2.3 システム・ディスケット

システム・ディスケットとは，〝CP／M システム〟が記録されており，そのディスケットをドライブAに挿入し，リセット・ボタンや IPL ボタンを押すことにより CP／M を起動できるディスケットのことを言います．

ここでは CP／M の基礎知識として，このシステム・ディスケット上にどのように CP／M が記録されているのか，そのフォーマットについて解説しましょう．

ディスクについては，8 インチ片面単密度の標準ディスケットを代表して取り上げますが，両面倍密度やミニフロッピー・ディスクの場合も，ディスクの両サイドを使ったり，トラックやセクタの数が増えたり減ったりするだけで，基本的には全く同じ考え方です．

Figure-2.3.1 にシステム・ディスケットのフォーマットを示します．

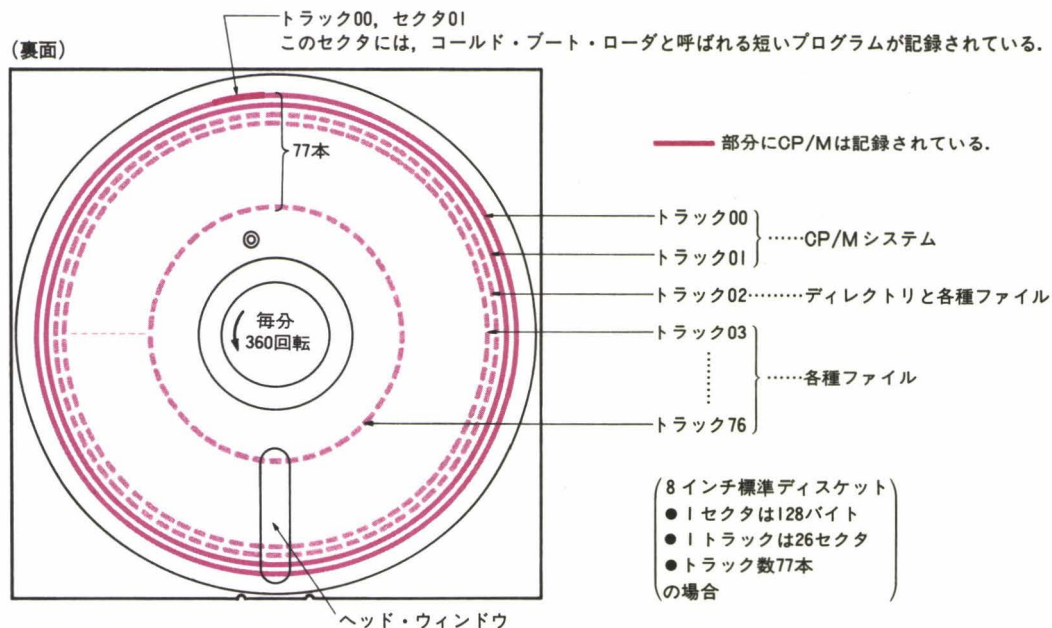



Figure-2.3.1 CP／Mシステム・ディスケットの記録状態

図はディスケットの裏面（ラベルの貼ってない側）を示しています．ディスクは矢印の向きに毎分360回転し，リード／ライト・ヘッドは紙面に垂直に長円形のヘッド・ウィンドウを通してディスク表面に接触します．実際の動作は，ヘッド自身が動くのではなく，ヘッドはディスク表面からわずかに離れて固定されており，リード／ライトを行う時に，ヘッドの反対側からヘッド・ロード・アームに付いているフェルトのパットで，ポリエステル・フィルムのやわらかいディスクをヘッドに押し付けます．このことを〝ヘッド・ロード〟と言います．

また，両面のミニフロッピー・ディスクの場合は，ディスケットを挿入してフタを閉じた時点で，両サイドともヘッドがディスク表面に接触する(ロード状態)構造のものもあります．その場合には，電源が入っていれば常にディスクが回転しているのではなく，ディスクがアクセスされるごとに，回転・停止をくり返します．

さて，図の77本のトラックの内，最外周の２本（トラック00と01）にCP／Mシステムは記録されています．このトラックのことを〝システム・トラック〟と呼びます．トラック２本分のメモリ容量は，128×26×２＝6656バイトであり，標準CP／Mディスクのフォーマットを採用する場合には，BIOS部を含むCP／M全システムは約6.5Kバイト以内に収める必要があります．但し，ディスケットの互換性を考慮しなければ，何も２本のトラックに無理して詰め込む必要はなく，システム・トラックの本数は任意に設定可能であり（CP／Mに付属の〝DISKDEF．LIB〟という特別のプログラムを利用して行う），BIOSをユーザーの仕様によっては，さらに大きく拡張することもできます(標準ディスケットは，その互換性こそ，第１の存在理由なので，誰もそんなことはしませんが)．

２本のシステム・トラックの１番最初のセクタ（トラック00，セクタ01）には，128バイト以下のコールド・ブート・ローダ（コールド・スタート・ローダなどとも呼ぶ）と呼ばれる特別なプログラムが入っています．このコールド・ブート・ローダは，２本のシステム・トラック全体を読み出すと同時に，その読み出されたCP／Mシステムをメモリ上の所定のアドレスにロードする重要な働きをします．

２本のシステム・トラックに続く３番目のトラック（トラック02）も特別のトラックであり，ディスク上の各ファイルの住所録とでも言うべき〝ディレクトリ〟の格納トラックになっています．８インチの標準ディスケットの場合，ディレクトリ・エリアにはこのトラック02の内，16のセクタを当てています．

スキュー（セクタの飛び越し読み書き）

たいていのDOSがそうであるように，CP／Mも回転しているディクス上のデータを，少しでも高速にリード／ライトできるように，セクタの飛び越し読み書き（〝スキュー〟と呼ぶ）を行っています．もし，〝スキュー〟がなく，セクタを１，２，３……と順序通りリード／ライトする場合には，セクタ１をリード／ライトして，そのデータをコンピュータ内部で処理し，次のセクタ２をリード／ライトしようとしても，セクタ２はすでに通り過ぎており，もう１回転待たなければなりません．

これでは，ディスク1回転につき，1セクタしかリード／ライトすることができず，ディスク・アクセスのスピードはたいへん遅くなってしまいます．

これを解決する手段として，1セクタのリード／ライトにおけるコンピュータ内部での処理時間分のセクタの回転を見送り，見送った次に来るセクタを，ロジカル的な next セクタとすれば，1回転に何セクタもリード／ライトすることができ，最大のパフォーマンスが得られることになります．8インチの標準ディスケットの場合，このスキュー・ファクタは6であり，6セクタごとに飛び飛びにリード／ライトされています．よってトラック02のディレクトリ・エリアもこのスキュー通りに記録され，具体的にディレクトリが記録されるセクタは，

1，7，13，19，25，5，11，17，23，3，9，15，21，2，8，14

の順に計16セクタが当てられています(21→2のように，6セクタごとでない箇所もあります)．実際にこのトラック02の，セクタ01と07をユーティリティー・プログラムのセクタ・ディスプレイでダンプしたものが，第3章「DIR コマンド」の項の Figure-3.1.16 に示してありますので参照して下さい．

ディレクトリ・トラックであるトラック02の，ディレクトリとして使用する16のセクタ以外のセクタと，最終トラックであるトラック76までは，ユーザー・ファイルの格納エリアであり，CP／M の各種トランジェント・コマンド・ファイルやユーザーの各種ファイルなどが記録されます．

以上述べた8インチの標準ディスケットのフォーマット一覧表をFigure-2.3.2 に示します．

(8インチ片面単密度，77トラック／ディスク，26セクタ／トラック，128バイト／セクタ)

トラック#	セクタ#	用途
00	01	コールド・ブート(コールド・スタート)・ローダ
	02～17	CCP——Console Command Processor
	18～26	BDOS——Basic Disk Operating System
01	01～19	
	20～26	BIOS——Basic Input Output System
02	1,7,13,19,25,5,11,17,23,3,9,15,21,2,8,14	ディレクトリ
	上記以外	ユーザー・ファイル
03～76	全部	

(トラック00～01：CP/Mシステム)

Figure-2.3.2 8インチ標準ディスケットのフォーマット

2.4 CP/M起動のメカニズム

CP/Mを起動するのは非常に簡単です．システム・ディスケットをドライブA：に挿入し，リセット・ボタン（あるいはIPLボタンなど）を押すことにより，直ちにディスクのシステム・トラックに格納されているCP/Mシステムが読み出され，メモリにロードされCP/Mが起動します．

日常，当り前のこととして何気なく行っているこの〝起動〟のメカニズムを理解することは，これから行う，すべてのCP/Mコマンドの実習を始める前知識としても意味のあることです．

では，リセット・ボタンなどでCP/Mが起動する，一般的なCP/Mマシンの場合について解説しましょう．

起動のメカニズムは大きく分けてFigure-2.4.1～2.4.3に図示する，3つのステップが連続して実行されます．

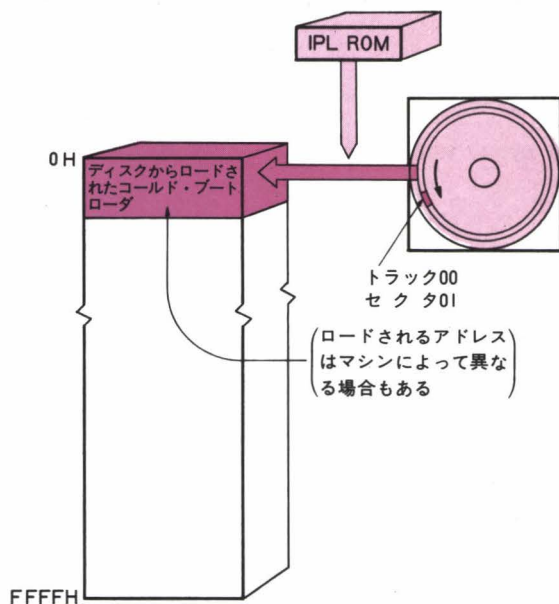



Figure-2.4.1 起動のメカニズム(STEP 1)

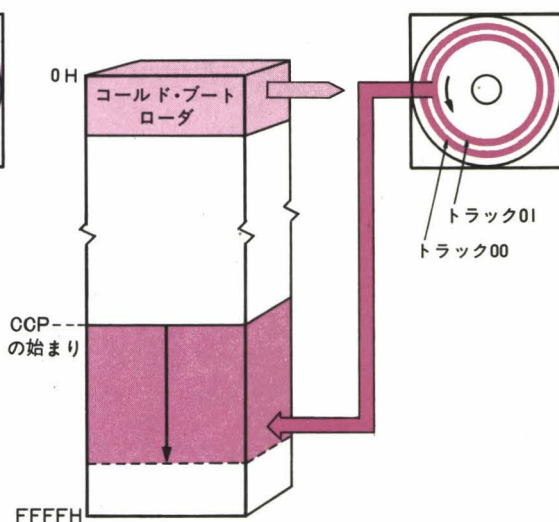



Figure-2.4.2 起動のメカニズム(STEP 2)

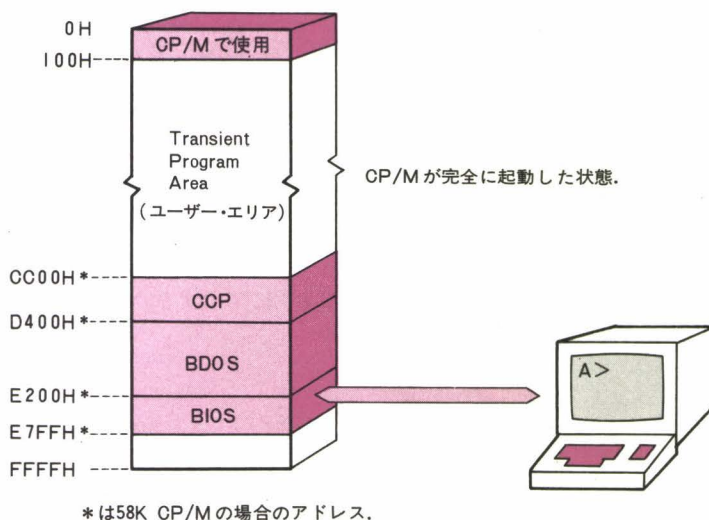
（STEP 3 ）
CP/Mシステム全体のロードが終了すると，コントロールはコールド・ブート・ローダからCP/Mに移り，CP/Mが完全に起動する．

CP/Mが完全に起動すると，BIOSを通してコンソールにプロンプト〝A>〟が出力され，キー入力待ちになる．

Figure-2.4.3　起動のメカニズム（STEP 3）

STEP1

システム・ディスケットをドライブAに挿入し，リセット・ボタンを押すと，その時に限りアクティブ（常時はメイン・メモリのアドレス空間には存在しない――存在しているマシンもあるが――）になるIPL（イニシャル・プログラム・ローダ）が働き，システム・ディスケット上のトラック00，セクタ01の128バイトのデータを読み出し，アドレス0Hからメイン・メモリにロードします（このロード・アドレスは，コンピュータによって異なります．例えばPC-8001 CP／Mの場合は，C000Hから）．この128バイトには，〝コールド・ブート・ローダ〟と呼ばれる短いプログラムが書かれており，これがCP／Mを起動させるためのメイン・プログラムになるのです．

STEP2

トラック00，セクタ01の1セクタに収まっていたプログラムをメイン・メモリにロードし，制御をそのプログラムであるコールド・ブート・ローダに渡せば，IPLはディスエーブルになり，メモリ空間には存在しなくなります（常駐しているマシンもある）．

IPLから制御を引き継いだコールド・ブート・ローダは，直ちに自分自身の入っていたセクタを除く，トラック00と01の2本のトラック全体を読み出し，所定のアドレスにロードする作業を始めます．CP／Mのメモリ上の始発点であるCCPの先頭アドレスから順に，メイン・メモリにロードして行き，2本のシステム・トラック全体を無事にロードし終ったことを確認すると，コールド・ブート・ローダは，BIOS部の先頭に位置する〝BOOT〟のエントリ・ポイント（第4章Figure-4.10.8参照）にとび込んで，その役目を終ります．

STEP3

BIOSの先頭のエントリ・ポイント〝BOOT〟に引き継がれた制御は，スクリーンのオール・クリア，CP／Mオープニング・メッセージの出力，それに各I／Oポートのイニシャライズなどを行ない，Figure-2.5.2に示すアドレス0Hのウォーム・ブートと，05HのBDOSへエントリするための各ベクトルを書き込んだ後，CCPへジャンプします．

CCPは，コンソールへCP／Mプロンプト〝A＞〟を出力をしたのちキーボードのスキャンを開始します．これで完全にCP／Mが起動し，すべてのコマンドを受け付ける準備が整ったわけです．

2.5 アドレス0H〜FFHのCP/Mが使用するエリアについて

CP／Mのユーザー・エリア（TPA）は，Figure-2.4.3にも示したように，アドレス100Hから始まっています．では一体0HからFFHまでの256バイトは何に使われているのでしょう．

この256バイトは，〝システム・エリア〟とか〝システム・スクラッチ・エリア〟などと呼ばれており，CP／Mが働くために，なくてはならないワーク・エリアになっています．

その内部の様子をFigure-2.5.1に示します．この中で色が付いている部分が，CP／Mの動作上なくてはならない命令やデータやバッファであり，その他の部分は現在のCP／M（Version 2.2）では使われていません．必要であれば，ユーザー・プログラムで使用してもかまいません．

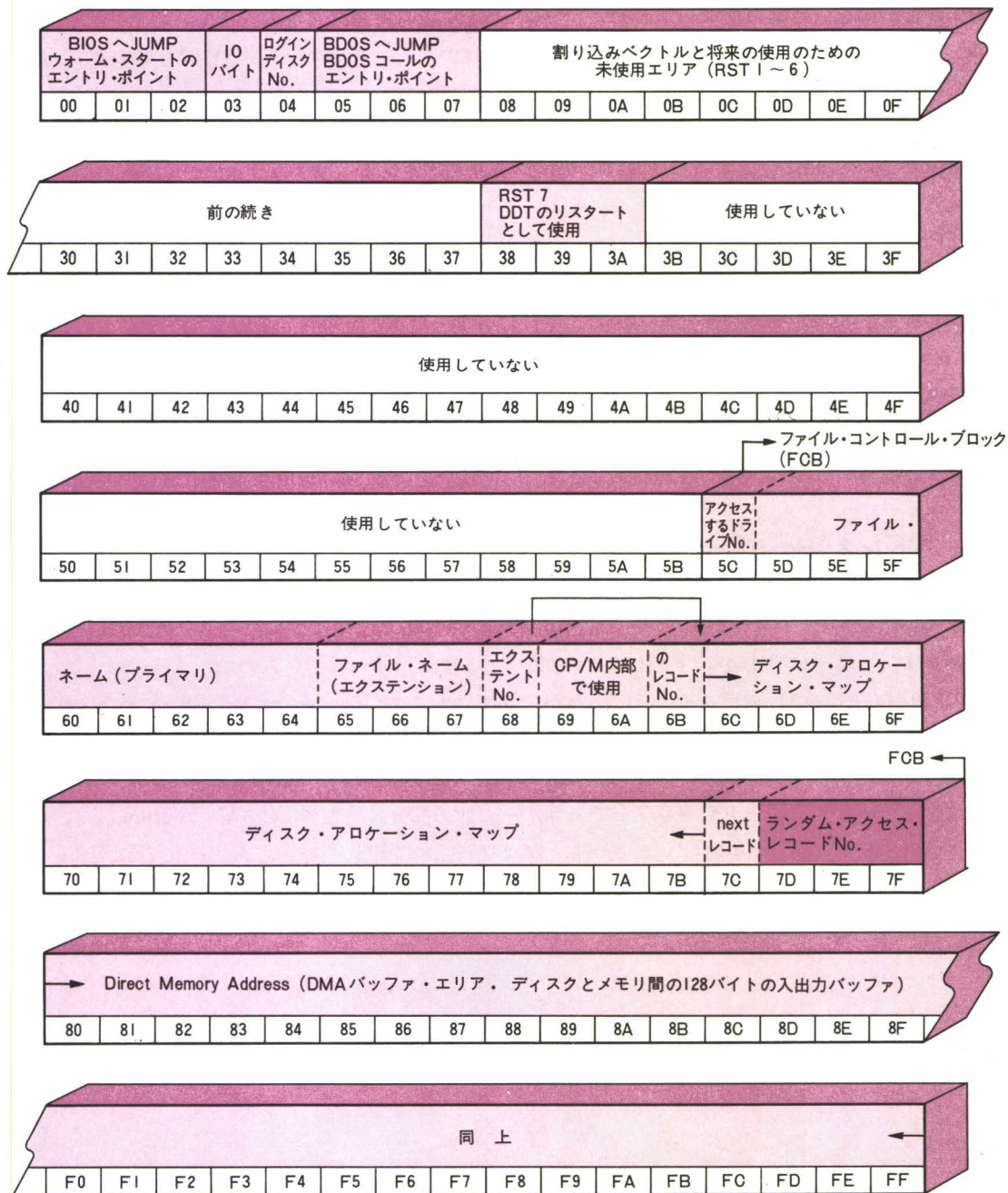



Figure-2.5.1 CP／Mが使用するスクラッチ・エリア(0H～FFH)のメモリマップ

もう少し詳しく解説しましょう．参考のために，CP／M が起動した直後の 0000H ～ 00FFH のメモリの内容を Figure-2.5.2 にダンプリストとして示します．

CP/Mの起動の際，初期設定されるのはこの部分だけである．
その他の部分のデータは意味はない．

コールド・ブート・ローダの残りゴミ．

```
0000  C3 03 BA 00 00 C3 06 AC 16 33 0E 02 06 04 79 CD  .........3....y.
0010  2A 00 15 CA 00 BA 06 00 0C 79 FE 1B DA 0F 00 3E  *........y.....>
0020  53 D3 E8 DB EC 0E 01 C3 0C 00 D3 EA CD 41 00 3E  S............A.>
0030  88 B0 D3 E8 DB EC B7 F2 41 00 DB EB 77 23 C3 34  ........A...w#.4
0040  00 DB E8 E6 9D C8 1D C2 02 00 32 80 00 2F D3 FF  ..........2../..
0050  C3 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .P..............
0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 EA F8  ................
0070  E2 F8 02 32 32 00 0A 00 50 BD 91 BB 31 BD D6 BA  ...22...P...1...
0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00F0  00 00 00 00 00 00 00 00 00 00 00 00 2F 00 12 00  ............/...
```

00H～02H（ウォーム・スタート・ベクトル）

この 3 バイトは，ウォーム・スタートの入口へ導く，BIOS 部先頭のジャンプ・ベクトルへのジャンプ命令が書き込まれています（Figure-4.10.8 の BIOS ジャンプ・ベクトル付近のリストを参照．ラベル〝WBOOT:〞がそれに当る）．

CP／M のアセンブラや DDT などが終了して CP／M にもどる時や，キーボードから Ctrl-C をキーインした時などは，まずこのアドレス 0000H にジャンプして来ます．そして，ここから BIOS の先頭より 2 番目に位置するウォーム・スタートのエントリ・ポイントへジャンプして行き，ウォーム・スタートが起ります．

ジャンプ命令のインストラクション・コードである〝C3H〞に続く 2 バイトは，ジャンプ先アドレスであり，この値は CP／M のサイズにより変化します．CP／M 上で走る応用プログラムを作成する場合，プログラムの終了には〝JMP　0H〞を実行させ，リブートさせて CP／M にコントロールをもどすのが一番安全確実な方法です．

03H（IO バイト）

IO バイトと呼ばれ，CP／M がディスク装置以外の周辺装置を扱う時に，その割り付け（4 つのロジカル・デバイスに対する，それぞれのフィジカル・デバイスの割り付け）や選択に，この 1 バイトを〝チャネル・スイッチ〞として使用します．1 バイトの 8 ビ

ットを，2ビットずつ4つのロジカル・デバイスに割り当てて，それぞれの2ビットの組み合わせで，4通りのフィジカル・デバイスの指定が可能です．この詳細については第1章の「IO バイトについて」の項で説明していますので参照ください．

04H（ログイン・ディスク No.）

ログイン・ディスク(Currently logged Disk)の番号を保持しています．A：=00H, B：=01H, C：=02H, …P：=0FH で表されます．Figure-2.5.2 では，起動直後の〝A＞〞の状態ですから，値は 00 となっています．

05H～07H（BDOS エントリ・ベクトル）

システム・コールのための BDOS へのエントリ・ポイントであり，ファンクション No. とパラメータを各レジスタに持って，このアドレス 05H を〝CALL〞することにより，それぞれのファンクションが実行されます．また，05H の JMP インストラクション・コード〝C3H〞に続くジャンプ先アドレスは，CP/M のサイズによって変化し，このアドレス値が BDOS の最下位アドレス＋6H であり，ユーザー・プログラムで使用可能な最高アドレス＋7H を表しています．但し，DDT などのトランジェント・プログラムが CCP 部にオーバーレイ（重ね書き）される場合は，このジャンプ先アドレスが，適当な中継アドレス（最終的には BDOS へ導く）に書き替えられます．DDT を起動した場合の 00H ～ FFH 間のメモリ・ダンプリスト，Figure-2.5.3 と Figure-2.5.2 を比較してみて下さい．

```
0000 C3 03 BA 00 00 C3 00 9C 16 33 0E 02 06 04 79 CD ..........3....y.
0010 2A 00 15 CA 00 BA 06 00 0C 79 FE 1B DA 0F 00 3E *.........y.....>
0020 53 D3 E8 DB EC 0E 01 C3 0C 00 D3 EA CD 41 00 3E S.............A.>
0030 88 B0 D3 E8 DB EC B7 F2 C3 86 A2 EB 77 23 C3 34 .............w#.4
0040 00 DB E8 E6 9D C8 1D C2 02 00 32 80 00 2F D3 FF ...........2../..
0050 C3 50 00 00 00 00 00 00 00 00 00 00 00 44 55 4D .P............DUM
0060 50 20 20 20 20 43 4F 4D 00 00 80 04 68 00 00 00 P    COM....h...
0070 00 00 00 00 00 00 00 00 00 00 00 00 04 BD D6 BA ................
0080 10 D8 C3 2A 05 CD 41 00 CA 2A 05 11 00 00 01 00 ...*..A..*......
0090 00 21 8B 06 09 7E FE 20 CA B0 00 CD 7B 00 6B 62 .!...~. ....x.kb
00A0 29 29 29 29 5F 16 00 19 EB 03 79 FE 04 C2 91 00 ))))_......y.....
00B0 42 4B 7B 05 04 C9 CD 85 00 C2 2A 05 C9 21 05 00 BK{........*..!..
00C0 36 C3 36 C3 21 00 00 22 06 00 C9 17 17 17 E6 38 6.6.!..".......8
00D0 C9 17 17 17 17 E6 30 C9 EB 2A 8B 06 EB 7B BE C2 .......0..*...{..
00E0 E7 00 23 7A BE C8 2B 2B 2B 0D C2 DD 00 0D C9 06 ..#z..+++.......
00F0 04 D5 11 8B 06 1A BE C2 02 01 23 13 05 C2 F5 00 ..........#.....
```


Figure-2.5.3　DDT起動後の0H～FFHのメモリ内容

38H～3AH（DDT でのリスタート・ベクトル）

DDT で，デバッグ中のプログラムからコントロールを DDT にもどすための，〝RST 7〟のベクトルが書き込まれます．この使い方は「DDT コマンド」のところで解説していますので参照して下さい．この〝RST 7〟のためのベクトルは，CP／M によって書き込まれるのではなく，DDT が起動した時点で DDT が書き込みます．Figure-2.5.3 と Figure-2.5.2 を比較して下さい．

また，このジャンプ先アドレスが CP／M のサイズによって変化するのは，ほかのものと同様です．

5CH～7CH（FCB：ファイル・コントロール・ブロック）

CP／M　DOS のファイルに関するすべての働きは，この FCB を通して行われます．

FCB の考え方は，CP／M に限らず，細部の違いこそあれ，ほとんどの DOS に用いられているもので，ディスク上のファイル処理・管理に必要な各種のパラメータを1つのブロックとして持たせるものです．Figure-2.5.3 に示されている，DDT で〝DUMP．COM〟を TPA にロードした場合の FCB アドレスの内容に注目して下さい．〝DUMP．COM〟がその中に書かれていることが分かります．FCB に関する知識はディスクのアクセスを含むプログラムを開発する上で必要です．この FCB に関しての詳細は続巻〝応用 CP／M〟で，〝システム・コール〟に関連して解説を行います．

（アドレス 5CH ～ 7CH はデフォルト値であり，任意のアドレスに設定可能．）

7DH～7FH（ランダム・レコード・ポジション）

ランダム・アクセスを行う場合のランダム・レコード No. とそのディスク上のポジションを示します．最初の2バイトが0～65535のレコード No. を表します．これらの部分は Version 1.4 にはありません．

（アドレス 7DH ～ 7FH はデフォルト値であり，任意のアドレスに設定可能．）

80H～FFH（DMA バッファ・エリア）

DMA（Direct Memory Address または，Disk Memory Access）バッファと呼ばれる**128**バイトのディスクの**Input** および **Output** 用のバッファです．ディスクへの読み書きは，必ずこのバッファを通して行われます．

Figure-2.5.3 のアドレス 80H ～ FFH の内容と，〝DUMP．COM〟をダンプした最後の128バイトとを比較して下さい．コマンド〝DDT　DUMP．COM〟によるディスク・アクセスの一番最後（DDT が起動し，それによってファイル〝DUMP．COM〟を TPA に読み込んだ最後のディスク・リードによる128バイト）のデータが，この DMA バッファに残っていることがわかります．

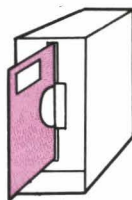
（アドレス 80H ～ FFH はデフォルト値であり，任意のアドレスに設定可能．）

以上がアドレス 0H～ FFH 間の CP／M システム・スクラッチ・エリアと呼ばれる部分の概略です．この部分は CP／M での実行を目的とする，アセンブラによるソフトウェア開発に密接に関係する部分であり，この辺の知識が絶対的に必要になります．また，続巻で解説するシステム・コールとも密接に関連しています．しかし，CP／M 上で各種高級言語やアプリケーション・ソフトを実行する上では，特に知る必要はありません．その場合，CP／M は各種ソフトウェアを走らせるための，単なる縁の下の OS にすぎず，これらの面倒なことはなるべく我々の目につかず，透明な方が良く出来た OS と言えるのです．

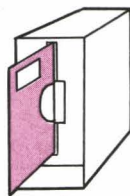
実習のはじめに

これから色々なコマンドを通してCP/Mの機能を解説して行きますが，その前に次の事をよく頭に入れておいて下さい．

★実習に使用する2枚のディスケットについて

実習は原則としてこの2枚のディスケットで行います．

ドライブA：には

```
A>DIR ┘
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

ドライブB：には

```
A>DIR B: ┘
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
A>
```

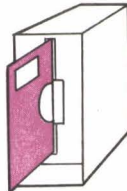
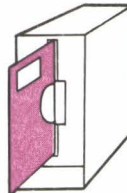
★ログイン・ディスク，プログラム・ディスクとアクセスされるディスクについて

CP／M を使いこなすには，現在のログイン・ディスクと，実行しようとするプログラム(コマンド)が存在するディスク，それに，そのプログラムによりアクセスされるディスクの三者の関係を，コマンド・ラインに自由に記述できなければなりません．

慣れないうちは少し戸惑いますが，原則は極めて単純なことなのです．

　ファイル名の頭には，必ずドライブ名（A：，B：，…）を付ける．
　但し，ドライブ名がログイン・ディスクと同一の時は省略してよい．

ただこれだけのことです．つまり，実行しようとするプログラム（コマンド）名の頭にドライブ名を付け，そのプログラムがアクセスするファイル名の頭にも，ドライブ名を付ければよいのです．

では，4章で紹介する STAT コマンドを使って，これら三者のいろいろなケースを実行してみますので，その実行例から三者の関係を学んで下さい．ログイン・ディスク名を表す〝A>〞や〝B>〞とファイル名の頭に付ける（あるいは省略されている）ドライブ名に注目して下さい．

```
A>STAT M80.COM ↲
File Not Found -------ドライブA：には〝M80.COM〞がない．

A>STAT B:M80.COM ↲
 Recs  Bytes  Ext Acc
  137    18k    2 R/W B:M80.COM ------ドライブB：上にあった．
Bytes Remaining On B: 81k

A>STAT ASM.COM ↲
 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM ------〝ASM.COM〞はドライブA：上にある．
Bytes Remaining On A: 74k
```

```
A>STAT B:ASM.COM ↵
File Not Found ----- ドライブB：上に "ASM.COM" はない.

A>B:↵ --------- ログイン・ディスクをB：にチェンジ.

B>STAT M80.COM ↵
STAT? ---------ドライブB：にはSTATコマンドである "STAT.COM" がない.

B>A:STAT M80.COM ↵
 Recs  Bytes  Ext Acc
  137    18k    2 R/W B:M80.COM ----- ドライブA：上のSTATコマンドを実行.
Bytes Remaining On B: 81k                "M80.COM" はドライブB：上にある.

B>A:STAT ASM.COM ↵
File Not Found ----- "ASM.COM" はドライブB：上にはない.

B>A:STAT A:ASM.COM ↵
 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM ------ドライブA：上に "ASM.COM" がある.
Bytes Remaining On A: 74k

B>A: ↵ ---------ログイン・ディスクをA：にチェンジ.

A>
```

★コマンド一般形で使用する記号について

コマンドの一般形で使用する記号の意味は次の通りです.

記号	x:, d:, s:,など	filename.ext	filematch	∧Cなど	␣	↵
意味	ドライブ名 A～Pまでのドライブ名（実際にドライブが接続されているもの）を指定する.	ファイル名 「filename.ext」の時はフルネーム,「filename」の時はプライマリ・ネームのみを使用する.	ファイル・マッチ "*"は, プライマリ・ネームまたはエクステンションの"すべて"を表し, "?"は,その位置の"すべての1文字"を表す.	Ctrl-Cなどのコントロール・キャラクタ.	スペース	キャリッジ・リターン
実際の例	A: B: C: ⋮ P:	DUMP.ASM ED.COM ABCDEFGH.123 12AB.Z0 ABCD 123 など.	*.COM TEST.* A??D.X?Z ??12.* *.* など.	それぞれのコントロール・キャラクタをキーインする.	スペースをキーインする.	キャリッジ・リターンをキーインする.

★本書のリストに使用する文字の字体について

本書では，誤植などを避けるため，リスト類はすべてコンピュータのプリンタ出力をそのまま使用していますが，O（オー）と0（ゼロ），l（エル）と1（イチ）など紛らわしいものもありますので，ご注意下さい．

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
```

★コントロール・キーによるライン・エディッティング機能について

コマンドをキー入力する際や，エディタ（ED）や DDT 内で文字列やサブ・コマンドをキー入力する際に，いくつかのライン編集機能が働きます．また，次々と表示されて行くスクリーン（コンソール）上の表示を一時ポーズ（フリーズ）状態にしたり，スクリーンに表示されるものを同時にプリンタ（リスト・デバイス）へ出力させることもできます．

これらの機能はキーボードからのコントロール・キャラクタによるコマンドで働き，次に示す合計10種類ほどのコマンドがあります．これらは Ctrl キーと共に用います．

1文字デリート

DEL/RUB 又は Back space	最後に入力した文字を1文字づつ削除する．削除された文字はマシンにより，スクリーン上から消去されるものと，再表示（エコーバック）されるものとがある．
Ctrl-H	同上．但し削除された文字はスクリーン上からも消去される．version 1.4ではこの機能なし．

1ライン・キャンセル

Ctrl-U	入力した1行を全部キャンセルし，キャンセルした行の最後に〝キャンセルされた〞という意味の〝#〞記号を表示して，カーソルを次の行の先頭にセットする．
Ctrl-X	同上．但しキャンセルされた行は，スクリーン上の表示も同時に消去される．カーソルは同じ行の先頭にもどる．version 1.4では Ctrl-U と同じ機能．

スクリーン・コントロール

Ctrl-S	スクリーン表示がスクロールしている時など，一時ポーズ（フリーズ）状態にする．再び Ctrl-S 又は他のキーを入力するとキャンセルされる．
Ctrl-R	入力された1行を次の行にきれいに〝清書〟して再表示する．1文字削除でエコーバックされた見づらい行を確認する場合などに使う（1文字削除においてエコーバックしないものもある）．
Ctrl-E	スクリーン上の表示のみに関する復帰・改行．このために入力されているコマンドが実行されることはない．

ライン・ターミネート

Ctrl-M	Carriage Return と同じ．
Ctrl-J	Line Feed と同じ．version 2.0 以上は Carriage Return の代用として使用可．

Ctrl-I	8文字ごとのタブ機能
Ctrl-P	スクリーンに表示されるものを〝LST：〟デバイス（通常はプリンタ）にも同時出力する．プリンタが動作可能な状態でないと，そこで hang-up(立ち往生)してしまうので注意すること．Ctrl-Pは入力のたびに ON-OFF をくり返すトグル動作になっている．
Ctrl-C	リブート（ウォーム・スタート，ウォーム・ブート）を起す．コマンド・ラインの最初に入力した時のみ有効．
Ctrl-Z	ED 内において，インサート・モードを終了する．また，サーチや置き換えのターミネータとして使用する．PIP のターミネータとして使用する．

★CP/Mのエラーについて

CP／M を運用していると，［BDOS ERR ON A: R／O］とか，［NO SPACE］とか，いろいろな場合にいろいろなエラー・メッセージが表示されることがあります．そのほとんどは，ユーザー側の問題であり，CP／M 自身がおかしかったり，ハードウェアのトラブルなどのエラーは，まず発生しません．もしそのようなエラーがしばしば発生するようであれば，ハードウェアの欠陥か，BIOS のソフトウェア的な欠陥があるのです．コンピュータ・システム全体のチェックを行う必要があります．

CP／M がエラーを検出して出力するエラー・メッセージの種類は，CP／M に共通なものと，各コマンドやプログラムに固有のものとがありますが，エラーの原因は，通常次に示すような単純なものがそのほとんどですから，適切な処理を行えば，解決するでしょう．

○ディスクの全メモリ容量をオーバーして書き込みを行った．
○ディスクの収容可能なファイル数をオーバーして，ファイルを作成した．
○ファイル名の指定やドライブ名の指定を誤った．
○ディスケットを交換した後，Ctrl-C を行わずに書き込みを行った．
○"R／O" のファイルに削除や書き込みを行った．
○未フォーマット（あるいは異なるフォーマット）のディスケットを使用した．
○ディスク上のデータが何らかの原因で破壊されていた．
○損傷しているディスケットを使用した．
など．

CP/Mがエラーを検出して，処理がストップした場合，通常はCtrl-Cをキーインして，今までの処理をキャンセルし，CP/Mに戻します．しかしCtrl-C以外のキーインを行うと，エラーの種類によっては，エラーを無視して，エラーのままで処理を続行できるものもあります．この場合は後で必ず，処理されたデータをチェックしなければなりません．

3章　ビルトイン・コマンド徹底実習

3.1　DIR　(ファイル名リストアウト・コマンド)

——DIRectory コマンド——

コマンド形式・機能

1　**DIR␣x:**
　　ドライブx：上のすべてのファイル名をリストアウトする.

2　**DIR␣x:filename.ext**
　　ドライブx：上のファイル名〝filename.ext〟を捜し，存在していればそのファイル名をリストアウトする.

3　**DIR␣x:filematch**
　　ドライブx：上のファイル名からファイル・マッチしたものをすべてリストアウトする.

注1)　x：はそれがログイン・ディスクの場合，省略できる.

注2)　〝SYS〟アトリビュートの付いているファイルは，いずれの場合もリストアウトされない.

実習1　DIR␣x:

ログイン・ディスクがA：の場合.

```
A>DIR
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

Figure-3.1.1　ドライブA：上のすべてのファイル名をリストアウト.

ログイン・ディスクがA：の時，ドライブB：上のすべてのファイル名をリストアウトする．

```
A>DIR B:
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
A>
```

Figure-3.1.2　ログイン・ディスクはA：のままで，ドライブB：上のすべてのファイルをリストアウト．

ログイン・ディスクをB：にチェンジして実行．

```
A>B:
B>DIR
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
B>
```

Figure-3.1.3　ログイン・ディスクをB：にチェンジしてから，ドライブB：上のすべてのファイルをリストアウト．Figure-3.1.2 と結果は同じ．

ログイン・ディスクがB：の時，ディスクA：上のすべてのファイル名をリストアウトする．

```
B>DIR A:
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
B>
```

Figure-3.1.4　Figure-3.1.1 と結果は同じ．

実習2 DIR␣x:filename.ext

ログイン・ディスクがA：の時，A：上の１つのファイルを確認する．

```
A>DIR DUMP.ASM
A: DUMP      ASM
A>
```

Figure-3.1.5
目的のファイルは存在していた．

```
A>DIR MBASIC.COM
NO FILE
A>
```

Figure-3.1.6
目的のファイルは存在しなかった．

ログイン・ディスクがB：の時，B：上の１つのファイルを確認する．

```
B>DIR MBASIC.COM
B: MBASIC    COM
B>
```

Figure-3.1.7
目的のファイルは存在していた．

ログイン・ディスクがA：の時，ドライブB：上の１つのファイルを確認する．

```
A>DIR B:MBASIC.COM
B: MBASIC    COM
A>
```

Figure-3.1.8
目的のファイルは存在していた．

```
A>DIR B:DUMP.ASM ↲
NO FILE
A>
```

Figure-3.1.9
目的のファイルは存在していなかった.

ログイン・デイスクがA：以外の場合の実習は，今までのものと同様なので省略.

実習3　DIR␣x:filematch

ログイン・ディスクがA：で，A：上のファイル名のエクステンションが ASM であるすべてのファイル名をリストアウトする場合.

```
A>DIR *.ASM ↲
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : PTBIOS48 ASM
A: PTBOOT48 ASM : DUMP     ASM
A>
```

Figure-3.1.10　プライマリ・ネームにファイル・マッチを使った例.

ログイン・ディスクがB：でB：上のファイル名のエクステンションが COM であるすべてのファイル名をリストアウトする場合.

```
B>DIR *.COM ↲
B: BASCOM   COM : M80      COM : L80      COM : MBASIC   COM
B>
```

Figure-3.1.11　上と同様に，プライマリ・ネームにファイル・マッチを使った例.

ログイン・ディスクがＡ：で，Ａ：上のファイル名に〝？〞と〝＊〞を使った場合．

```
A>DIR PT????48.* ♪
A: PTBIOS48 ASM : PTBOOT48 ASM
A>
```

Figure-3.1.12
エクステンションに〝＊〞，プライマリ・ネームに〝？〞を同時に使った場合．

「ＤＩＲ ＊．＊♪」は「ＤＩＲ♪」と同じです．

```
A>DIR *.* ♪
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

Figure-3.1.13　Figure-3.1.1と同じ結果です．

ログイン・ディスクがＡ：で，ドライブＢ：上のエクステンションがCOMであるすべてのファイル名をリストアウトする場合．

```
A>DIR B:*.COM ♪
B: BASCOM   COM : M80      COM : L80      COM : MBASIC   COM
A>
```

Figure-3.1.14　ログイン・ディスク以外のドライブにもファイル・マッチは使える．

ログイン・ディスクがＢ：で，ドライブＡ：上のファイルに〝＊〞と〝？〞を使った場合．

```
B>DIR A:PT????48.* ♪
A: PTBIOS48 ASM : PTBOOT48 ASM
B>
```

Figure-3.1.15
Figure-3.1.12と同一結果．

解説　DIR

DIRコマンドが実行されると，8インチの標準ディスケットの場合は，トラック02上のセクタ1，7，13，19，25，5，11，17，23，3，9，15，21，2，8，14（6セクタごとのスキューがかかっているので，このような順序になる，〝スキュー〟に関しては，第2章の2.3を参照）の計16セクタを占めるディレクトリ・エリアを読み出し，目的のファイル名をコンソールにリストアウトします．

このディレクトリが実際にどのように記録されているのか，最初の2つのセクタ（セクタ1と7）をセクタ・ディスプレイ・プログラム（シンクウェア・ラブズ社）で見てみましょう．

```
A>B:SECDIS↲

    ======   SECTOR DISPLAY PROGRAM  V1.5   ======
               copyright by SYNCWARE LABS

input disk name A - D  >A
input track# 00-76  >02
input sector# 01-26  >01
D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M
input   D,   N,   A,   X,   R,    >D

DISK=A  TRACK=02   SECTOR=01
A:00   00 4D 4F 56 43 50 4D 20 20 43 4F 4D 00 00 00 4C  .MOVCPM  COM...L
A:10   02 03 04 05 06 07 08 09 0A 0B 00 00 00 00 00 00  ................
A:20   00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A  .PIP     COM...:
A:30   0C 0D 0E 0F 10 11 12 13 00 00 00 00 00 00 00 00  ................
A:40   00 53 55 42 4D 49 54 20 20 43 4F 4D 00 00 00 0A  .SUBMIT  COM....
A:50   14 15 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
A:60   00 58 53 55 42 20 20 20 20 43 4F 4D 00 00 00 06  .XSUB    COM....
A:70   16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

        .
        .
        .
        .
        .
        .

DISK=A  TRACK=02   SECTOR=07
A:00   00 45 44 20 20 20 20 20 20 43 4F 4D 00 00 00 34  .ED      COM...4
A:10   17 18 19 1A 1B 1C 1D 00 00 00 00 00 00 00 00 00  ................
A:20   00 41 53 4D 20 20 20 20 20 43 4F 4D 00 00 00 40  .ASM     COM...@
A:30   1E 1F 20 21 22 23 24 25 00 00 00 00 00 00 00 00  .. !"#$%........
A:40   00 44 44 54 20 20 20 20 20 43 4F 4D 00 00 00 26  .DDT     COM...&
A:50   26 27 28 29 2A 00 00 00 00 00 00 00 00 00 00 00  &'()*...........
A:60   00 4C 4F 41 44 20 20 20 20 43 4F 4D 00 00 00 0E  .LOAD    COM....
A:70   2B 2C 00 00 00 00 00 00 00 00 00 00 00 00 00 00  +,..............

D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M
input   D,   N,   A,   X,   R,    >
```

Figure-3.1.16　ディレクトリが記録されているセクタを見る．

このように，〝DIR〞でリストアウトした場合と同じ順序で各ファイルの〝住所録〞が並んでいるのが分かります．1つのファイルに32バイトが割り当てられ，最初の16バイトがファイル名などのエリア，後の16バイトが実際にファイルがセーブされている，ディスク上の位置を示すディスク・アロケーション・マップとなっています．1つのファイルに対して32バイトを必要とするディレクトが，16セクタ＝16×128バイト＝64×32バイトのエリアを持っているので，計64個のファイルを1つのディスケットに収納することができるわけです（8インチ標準ディスケットの場合）．

ファイルに〝SYS〞アトリビュート（Systemアトリビュート）が付いている場合，そのファイルは，DIR コマンドではリストアウトされず，〝存在しない〞とみなされます．

アトリビュートについては STAT コマンドの項で詳しく解説しますが，ここで参考までに ASM, DDT, ED, LOAD の4つの各 COM ファイルに〝SYS〞アトリビュートが付いている場合の，STAT コマンドによるファイル名のリストアウトと，DIR コマンドによるリストアウトを示します．

STAT コマンドによるリストアウト

```
A>STAT *.* J

 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:(ASM.COM)  ←
   96    12k    1 R/W A:BIOS.ASM
   69     9k    1 R/W A:CBIOS.ASM
   38     5k    1 R/W A:(DDT.COM)  ←
   80    10k    1 R/W A:DEBLOCK.ASM
   49     7k    1 R/W A:DISKDEF.LIB
   33     5k    1 R/W A:DUMP.ASM
    4     1k    1 R/W A:DUMP.COM
   52     7k    1 R/W A:(ED.COM)  ←
   14     2k    1 R/W A:(LOAD.COM)  ←
   76    10k    1 R/W A:MOVCPM.COM
   58     8k    1 R/W A:PIP.COM
  222    28k    2 R/W A:PTBIOS48.ASM
   26     4k    1 R/W A:PTBOOT48.ASM
   68     9k    1 R/W A:PTCPM48.COM
   41     6k    1 R/W A:STAT.COM
   10     2k    1 R/W A:SUBMIT.COM
    8     1k    1 R/W A:SYSGEN.COM
    6     1k    1 R/W A:XSUB.COM
Bytes Remaining On A:  106k

A>
```

Figure-3.1.17　（　）の付いているファイル名に注目．

このように〝SYS〞アトリビュートの付いているファイルは（　）で表示されます．

DIR コマンドによるリストアウト

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

Figure-3.1.18 上の STAT コマンドで( )の付いていたファイルが，リストアウトされていないことに注目．

このように，DIR では〝SYS〃アトリビュートの付いたファイルはリストアウトされません．

3.2　TYPE　（ファイル内容タイプアウト・コマンド）

――TYPE out コマンド――

コマンド形式・機能

TYPE␣x : filename.ext

ドライブx：上のアスキー・ファイル〝filename.ext〞をタイプアウトする.

注）　x：はそれがログイン・ディスクの場合，省略できる.

実習　TYPE␣x:filename.ext

ログイン・ディスクがA：の時，A：上のファイル，〝DUMP．ASM〞をタイプアウトする.

```
A>TYPE DUMP.ASM↲

;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
      .
      .
      .
      .
      .
      .
;       FIXED MESSAGE AREA
SIGNON: DB      'FILE DUMP VERSION 1.4$'
OPNMSG: DB      CR,LF,'NO INPUT FILE PRESENT ON DISK$'

;       VARIABLE AREA
IBP:    DS      2       ;INPUT BUFFER POINTER
```

```
OLDSP:  DS      2       ;ENTRY SP VALUE FROM CCP
;
;       STACK AREA
        DS      64      ;RESERVE 32 LEVEL STACK
STKTOP:
;
        END

A>
```

Figure-3.2.1　DUMP プログラムのソース・ファイルをタイプアウトする．

ファイル全部をタイプアウトし終ると，このようにCP/Mにもどります．タイプアウトされている途中で，Ctrl-Sをキーインすると，その時点でタイプアウト動作はポーズ(フリーズ)状態になります．ポーズを解除しタイプアウトを続行するには，何らかのキーインを行います．

ログイン・ディスクがA：の時，B：上のファイル〝TESTPRO．BAS〞をタイプアウトする．

```
A>TYPE B:TESTPRO.BAS ↲

10 '============================================================
20 '
30 '          ITI  CS  UCS  FOR MOUSE
40 '
50 '============================================================
60 '
70 PRINT "===== DISCRIMINATED AVOIDANCE-SAME PROGRAM WITH 4 BOXES =====
80 PRINT
90 '
100 '***** INITIAL SET *****
110 '
120 DEFINT A-Z     'DEFAIN ALL VALIABLES TO INTEGER
130 '
140 'DISK FILE NAME DEFINE
150 INPUT "FILE NAME";FILENAMEIN$
160 INPUT "DATE";DATEIN$
170 '
180 'DISK FILE OPEN
190 OPEN "R",#1,FILENAMEIN$
200 '
210 '--- DATA STORE AREA DIM DEFINE ---
220 'DATA BYTE BIT ASSIGNMENT
230 'BIT7,BIT6 ---- 0 0 = NO LEVER RESPONSE
240 '     "      ---- 0 1 = RESPONSE AT CS
250 '     "      ---- 1 0 = RESPONSE AT UCS
260 'BIT5,4,3,2,1,0 ---- LEVER RESPONSE COUNTER AT ITI
270 '
280 DIM DATABOX(4,100)     'BOX1 - BOX4 DATA BUFFER
290 DIM DSKDATA$(4)
```

```
300 DSKDATA$(1)=STRING$(100,0)     'BOX1 DISK DATA BUFFER
310 DSKDATA$(2)=STRING$(100,0)     'BOX2        "
320 DSKDATA$(3)=STRING$(100,0)     'BOX3        "
330 DSKDATA$(4)=STRING$(100,0)     'BOX4        "

A>
```

Figure-3.2.2　ログイン・ディスク以外のドライブ上の，BASIC 言語のソース・プログラムをタイプアウトする.

このファイルは最後までタイプアウトされていません．タイプアウトの途中でCP／Mにもどっています．このようにタイプアウトの途中でブレークをかけ，TYPEコマンドを打ち切るには，ポーズ・コマンドであるCtrl-S以外の何らかのキーインを行えば，その時点でCP／Mにもどることができます．

ログイン・ディスクがA：以外のケースも同様なので省略．

ファイル名にファイル・マッチは使えません．

```
A>TYPE DUMP.* ↲
DUMP.*?

A>
```

Figure-3.2.3
ファイル・マッチを使うと，このようにエラーとなります．

解説　TYPE

TYPEコマンドは，任意のファイルを読み出し，その内容をそのままコンソールに送ります．コンソールは送られてきたデータをそのまま表示しますので，指定するファイルはアスキー・ファイル(文字ファイル)でなければ，コンソールは表示することができません．試しにPIP．COMとか，ED．COMなどのマシン語ファイルをタイプアウトしてみて下さい．メチャクチャの表示になったでしょう．

また，コンソールへの表示は，8文字ごとのタブ処理が行われますので，Figure-3.2.1では，ラベル，オペランド，コメントなどの欄の頭が揃っています．

ここで参考までにFigure-3.2.1でタイプアウトしたDUMP．ASMのファイルのデータを，直接16進で見てみましょう．16進表示部の右側は，そのアスキー・コードを文字に変換したものです．TYPEコマンドでの表示と比較して下さい．タブは〝09H〞，復帰改行は〝0DH〞，〝0AH〞であることが分り

ますね.

```
A>B:DUMPHA DUMP.ASM↲

     ======    FILE DUMP(HEX & ASCII) PROGRAM  V1.5    ======
                   copyright by  -SYNCWARE LABS-

0000  3B 09 46 49 4C 45 20 44 55 4D 50 20 50 52 4F 47  ;.FILE DUMP PROG
0010  52 41 4D 2C 20 52 45 41 44 53 20 41 4E 20 49 4E  RAM, READS AN IN
0020  50 55 54 20 46 49 4C 45 20 41 4E 44 20 50 52 49  PUT FILE AND PRI
0030  4E 54 53 20 49 4E 20 48 45 58 0D 0A 3B 0D 0A 3B  NTS IN HEX..;..;
0040  09 43 4F 50 59 52 49 47 48 54 20 28 43 29 20 31  .COPYRIGHT (C) 1
0050  39 37 35 2C 20 31 39 37 36 2C 20 31 39 37 37 2C  975, 1976, 1977,
0060  20 31 39 37 38 0D 0A 3B 09 44 49 47 49 54 41 4C   1978..;.DIGITAL
0070  20 52 45 53 45 41 52 43 48 0D 0A 3B 09 42 4F 58   RESEARCH..;.BOX

0080  20 35 37 39 2C 20 50 41 43 49 46 49 43 20 47 52   579, PACIFIC GR
0090  4F 56 45 0D 0A 3B 09 43 41 4C 49 46 4F 52 4E 49  OVE..;.CALIFORNI
00A0  41 2C 20 39 33 39 35 30 0D 0A 3B 0D 0A 09 4F 52  A, 93950..;...OR
00B0  47 09 31 30 30 48 0D 0A 42 44 4F 53 09 45 51 55  G.100H..BDOS.EQU
00C0  09 30 30 30 35 48 09 3B 44 4F 53 20 45 4E 54 52  .0005H.;DOS ENTR
00D0  59 20 50 4F 49 4E 54 0D 0A 43 4F 4E 53 09 45 51  Y POINT..CONS.EQ
00E0  55 09 31 09 3B 52 45 41 44 20 43 4F 4E 53 4F 4C  U.1.;READ CONSOL
00F0  45 0D 0A 54 59 50 45 46 09 45 51 55 09 32 09 3B  E..TYPEF.EQU.2.;
      .
      .
      .
      .
      .
```

Figure-3.2.4　DUMPHA プログラムは CP/M には含まれていませんが，DDT コマンドでも同じように見ることができます.

3.3　REN　（ファイル名変更コマンド）

——REName コマンド——

コマンド形式・機能

REN␣x：新filename.ext＝旧filename.ext
同一ディスク：上で，〝旧filename.ext〞を〝新filename.ext〞に，ファイル名の変更を行う．
注）　x：はそれがログイン・ディスクの場合，省略できる．

実習　REN␣x:新filename.ext=旧filename.ext

ログイン・ディスク上のファイル名を変更する．

まず，REN コマンド実行前のログイン・ディスク上（現在はA：）の全部のファイル名を，DIR コマンドでリストアウトして，Figure-3.3.1 に示しておきます．

```
A>DIR ↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM ←— 注目
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

Figure-3.3.1　実習前のファル名の確認．

この中の右端の 3 行目のファイル〝BIOS．ASM〞を〝12345678．YXZ〞というファイル名にリネームしてみます．

```
A>REN 12345678.XYZ=BIOS.ASM ↲
A>
```

Figure-3.3.2
REN コマンドの実行．

REN コマンドが実行されました．再び DIR で全体のファイルを見てみましょう．

```
A>DIR
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : 12345678 XYZ ←― 注目
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

Figure-3.3.3　結果の確認．

このように〝12345678．XYZ〞にリネームされています．

ではこのファイル（内容は〝BIOS．ASM〞）を TYPE コマンドで，参考までにリストアウトしてみます．

```
A>TYPE 12345678.XYZ

;       MDS-800 I/O Drivers for CP/M 2.2
;       (four drive single density version)
;
;       Version 2.2 February, 1980
;
vers    equ     22      ;version 2.2
;
;       Copyright (c) 1980
;       Digital Research
;       Box 579, Pacific Grove
;       California, 93950
;
;
true    equ     0ffffh  ;value of "true"
false   equ     not true        ;"false"
test    equ     false   ;true if test bios
        .
        .
        .
        .
        .
```

Figure-3.3.4　中身は BIOS．ASM です．

このように当然ですが，中身は〝BIOS．ASM〞です．

ログイン・ディスク以外のドライブ上のファイル名を変更する.

ドライブB：上のファイル〝M80．COM〞（マイクロソフト社のマクロ・アセンブラ）を〝MACRO-80．COM〞と，ファイル名を変更しましょう．ログイン・ディスクはA：のままで行ってみます．

その前にドライブB：上の全部のファイル名を，DIR コマンドでタイプアウトして示しておきます．

```
A>DIR B: ↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
A>
```

注目

Figure-3.3.5　実行前のファイル名の確認．

では REN コマンドを実行します．

```
A>REN B:MACRO-80.COM=M80.COM ↲
A>
```

Figure-3.3.6
REN コマンドの実行．

この確認は後にして，ドライブB：上のもう1つのファイル〝L80．COM〞（マイクロソフト社のリンカー）を〝LINK-80．COM〞とファイル名を変更してみます．今回はログイン・ディスクをB：にチェンジしてから REN コマンドを実行します．

```
A>B: ↲
B>REN LINK-80.COM=L80.COM ↲
B>
```

Figure-3.3.7
ログイン・ディスクのチェンジ後，REN コマンドを実行．

REN コマンドが実行されました．REN に続くコマンド・ラインにFigure-3.3.6 にある〝B：〞がないことに注目して下さい．

さて，これでドライブB：上の2つのファイル名が変更されました．DIR コマンドで全体のファイルを見て確認してみましょう．

```
                                                    注目
B>DIR
B: BASLIB   REL : BASCOM   COM : MACRO-80 COM : LINK-80  COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
B>
```

Figure-3.3.8　結果の確認.

同じディスク上に，すでに存在するファイル名にはリネームできません.

ドライブA：上にはすでに，ファイル〝DEBLOCK．ASM〟があります．このディスク上のファイル〝DUMP．ASM〟を，〝DEBLOCK．ASM〟にリネームを試みます.

```
A>REN DEBLOCK.ASM=DUMP.ASM
FILE EXISTS
A>
```

Figure-3.3.9
すでに存在するファイル名へのリネームはできない.

このように「ファイルすでに在り」のメッセージが出力されて，リネームは不可能です.

新旧2つのファイルが，異なったディスク上に在る場合のリネームはできません.

```
A>REN A:ABCD.XYZ=B:MBASIC.COM
B:MBASIC.COM?

A>
```

Figure-3.3.10
同一ディスク上にないファイル名のリネームはできない.

このような2つのディスクにまたがる REN コマンドは受け付けられません.

ファイル・マッチは新・旧どちらのファイル名にも使えません.

```
A>REN *.OBJ=*.COM
*.OBJ=*.COM?

A>
```

Figure-3.3.11
ファイル・マッチは使えない.

すべてのCOMエクステンションのファイルを，OBJエクステンションにリネームしようと試みますが，このように不可能です．

解説　REN

RENコマンドは，変更しようとする旧ファイル名を，該当ディスクのディレクトリから捜し出し，もしそのファイル名が存在していれば，ディレクトリのそのファイル名の部分のみ，新ファイル名に書き替えます．あとは変化する部分は全くありません．

Figure-3.3.2のRENコマンドにより，ディレクトリが書き替えられる様子を，コマンドの実行前と実行後の該当ディレクトリを対比して示します．

```
DISK=A  TRACK=02   SECTOR=13
A:00  00 53 54 41 54 20 20 20 20 43 4F 4D 00 00 00 29  .STAT    COM...)
A:10  2D 2E 2F 30 31 32 00 00 00 00 00 00 00 00 00 00  -./012..........
A:20  00 53 59 53 47 45 4E 20 20 43 4F 4D 00 00 00 08  .SYSGEN  COM....
A:30  33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  3...............
A:40  00 44 55 4D 50 20 20 20 20 C3 4F 4D 00 00 00 04  .DUMP    .OM....
A:50  34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  4...............
A:60  00 42 49 4F 53 20 20 20 20 41 53 4D 00 00 00 60  .BIOS    ASM...    ←── 注目
A:70  35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 00 00 00 00  56789:;<=>?@....

D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M

                    〈RENコマンド実行〉

DISK=A  TRACK=02   SECTOR=13
A:00  00 53 54 41 54 20 20 20 20 43 4F 4D 00 00 00 29  .STAT    COM...)
A:10  2D 2E 2F 30 31 32 00 00 00 00 00 00 00 00 00 00  -./012..........
A:20  00 53 59 53 47 45 4E 20 20 43 4F 4D 00 00 00 08  .SYSGEN  COM....
A:30  33 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  3...............
A:40  00 44 55 4D 50 20 20 20 20 C3 4F 4D 00 00 00 04  .DUMP    .OM....
A:50  34 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  4...............
A:60  00 31 32 33 34 35 36 37 38 58 59 5A 00 00 00 60  .12345678XYZ...`   ←── 注目
A:70  35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 00 00 00 00  56789:;<=>?@....

D)isplay,   N)ext sector disp.,   A)uto next,   X) any sector   R)eboot CP/M
```

Figure-3.3.12　RENコマンド実行前・実行後のディレクトリの変化．

それぞれのトラック02，セクタ13のA：60で示される行に注目して下さい．ディレクトリのこの部分が，書き替えられていることが分かります．これがすなわち，リネームされたと言うことなのです．

3.4 ERA （ファイル削除コマンド）

——ERAse コマンド——

コマンド形式・機能

1 **ERA␣x : filename.ext**
　　ドライブx：上のファイル〝filename.ext〟を削除する.
2 **ERA␣x : filematch**
　　ドライブx：上のファイル・マッチするファイルをすべて削除する.
注1) 〝R／O〟アトリビュートの付いているファイルは，いずれの場合も削除できない.
注2) x：はそれがログイン・ディスクの場合，省略できる.

実習1 ERA␣x:filename.ext

ログイン・ディスク上のファイル〝filename . ext〟を削除する.

まず，ERA コマンド実行前の現在のログイン・ディスクA：上のファイルを DIR コマンドで示します.

```
A>DIR ↲                                   ← 注目
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

Figure-3.4.1 実行前のファイルの確認.

この中の最初のファイル〝MOVCPM . COM〟を削除してみます.

```
A>ERA MOVCPM.COM ↲
A>
```

Figure-3.4.2
1つのファイルを削除.

コマンドが実行されました．DIR で確認してみましょう．

```
A>DIR↲
A: PIP      COM : SUBMIT   COM : XSUB     COM : ED       COM
A: ASM      COM : DDT      COM : LOAD     COM : STAT     COM
A: SYSGEN   COM : DUMP     COM : BIOS     ASM : CBIOS    ASM
A: DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM : PTBOOT48 ASM
A: PTCPM48  COM : DUMP     ASM
A>
```

Figure-3.4.3　結果の確認.

ログイン・ディスク以外のドライブ上のファイルを削除する．

まず，ERA コマンド実行前のドライブB：上のファイルを，DIR コマンドで示します．

```
A>DIR B:↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
A>
```

注目

Figure-3.4.4　実行前のドライブB：上のファイルの確認.

このドライブB：上のファイル〝FORLIB．REL〞を，ログイン・ディスクはA：の状態で削除してみます．

```
A>ERA B:FORLIB.REL↲
A>
```

Figure-3.4.5
ログイン・ディスク以外のドライブ上のファイルの削除.

コマンドが実行されました．DIR で確認してみましょう．

```
A>DIR B: ↵
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : MBASIC   COM
A>
```

Figure-3.4.6　結果の確認．

実習2 ERA␣x:filematch

ログイン・ディスク上のすべての COM エクステンションを持つファイルを削除する．

```
A>ERA *.COM ↵
A>
```

Figure-3.4.7
ファイル・マッチを使った実行例．

コマンドが実行されました．DIR で確認してみましょう．Figure-3.4.3 と比較して下さい．

```
A>DIR ↵
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48 ASM : DUMP     ASM
A>
```

Figure-3.4.8　結果の確認．

ログイン・ディスク以外のディスク上のすべてのファイルを削除する．

ファイル・マッチ記号〝＊．＊〟で，すべてのファイルの削除ができます．但し事が重大なので，CP／M はコマンドの再確認を求めるメッセージを出力します．

```
A>ERA B:*.* ↲
ALL (Y/N)?Y ↲
A>
```

Figure-3.4.9
ドライブB：上のファイルのオール削除.

CP／M の問いに，〝Y″と答えてリターンすれば実行され，〝Y″以外の文字または何も入力せずリターンすれば，コマンドはキャンセルされて CP／M にもどります.

すべてのファイルを削除されたディスクB：を DIR で確認してみましょう.

```
A>DIR B: ↲
NO FILE
A>
```

Figure-3.4.10
結果の確認.

ログイン・ディスクを目的のファイルが存在するディスクにチェンジしてから ERA コマンドを実行する方が，〝うっかりミス″の防止上安全です.

例えば，Figure-3.4.5 の場合などは，次のように行う方が安全です．ディスク・チェンジしてから DIR で確認している点に注目.

```
A>B: ↲
B>DIR ↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
B>ERA FORLIB.REL ↲
B>
```

Figure-3.4.11　ERA コマンドを安全確実に実行する.

〝R／O″（リード・オンリー）アトリビュートの付いているファイルは削除できません.

現在，ドライブA：に挿入されているディスケット上のファイルの状態を，STAT コマンドで調べてみます.

```
A>B:STAT *.* ↵

 Recs  Bytes  Ext Acc
   96    12k    1 R/W A:BIOS.ASM
   69     9k    1 R/W A:CBIOS.ASM
   80    10k    1 R/W A:DEBLOCK.ASM
   49     7k    1 R/W A:DISKDEF.LIB
   33     5k    1 R/O A:DUMP.ASM   ←―注目
  222    28k    2 R/W A:PTBIOS48.ASM
   26     4k    1 R/W A:PTBOOT48.ASM
Bytes Remaining On A: 166k

A>
```

Figure-3.4.12 STAT コマンドによるファイルの確認.

このように，存在しているファイルの状態がリストアウトされました．ここでファイル〝DUMP．ASM〟のみ〝R／O〟と表示されていることに注目下さい．

この〝R／O〟すなわち〝書き込み・消去禁止〟のファイルにERAコマンドを実行してみましょう．

```
A>ERA DUMP.ASM ↵
Bdos Err On A: File R/O ^C
A>
```

Figure-3.4.13
R／O ファイルに対して，ERA コマンドを実行.

このように〝ファイルはR／Oである〟旨のメッセージが出力され削除できません．この状態ではCP／M にもどっていませんので，Ctrl-Cをキーインして CP／M にもどします．

〝R／O〟ファイルを削除するには〝R／O〟アトリビュートを〝R／W〟（リード・ライト）アトリビュートに変更してから ERA コマンドを実行します．

```
A>B:STAT DUMP.ASM $R/W ↵

DUMP.ASM set to R/W
A>
```

Figure-3.4.14
R／OファイルをR／Wファイルに変更.

詳しくは第4章「STATコマンド」で解説しますが，このように STAT コマンドでアトリビュートを変更します．これで〝R／W〟すなわち〝書き込み・消法可能〟ファイルに変更されたので，再度 ERA コマンドを実行してみます．

```
A>ERA DUMP.ASM ↲
A>
```

Figure-3.4.15
R／Wファイルに変更後の ERA コマンドの実行．

今度は実行されました．DIR で確認してみましょう．

```
A>DIR ↲
A: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB
A: PTBIOS48 ASM : PTBOOT48 ASM
A>
```

Figure-3.4.16　結果の確認．

解説　ERA

ERA コマンドも，カレント・ユーザー・エリア（現在のユーザー・エリア，第3章「USER コマンド」の項参照）内のファイルに対してのみ有効です．他のユーザー・エリアのファイルに対しては〝*．*〟を使った ALL ERASE コマンドを実行しても影響を与えません．

ERA コマンドを我々は普通，「ファイルを削除する」と言っていますが，物理的にはディスケット上からデータを〝消去〟するのではなく，該当ファイルのディレクトリの最初の1バイトを，〝E5〟(HEX)に書き替えるだけなのです．つまり，今まで〝有効〟であったファイルに，〝無効〟の印(E5H)を付けるのが ERA コマンドであると言えます．

その様子をA：，B：，2つのディスケットのデータを比較・ダンプするユーティリティー・プログラムである（シンクウェア・ラブズ社）〝CMPDISK〟で見てみましょう．最初は全く同じ2枚のディスケットで，A：が Figure-3.4.1 で示される内容の，元のディスケットであり，B：は Figure-3.4.2 の ERA コマンドを実行して，ファイル〝MOVCPM．COM〟を削除したディスケットです．

```
A>CMPDISK↲

      ======   DISK COMPARE PROGRAM  V1.5   ======
             copyright by  -SYNCWARE LABS-
 This program compare diskA: with diskB: by track sector to track sector.

INSERT SOURCE DISKET ON DRIVE  A:
       OBJECT DISKET ON DRIVE  B:

input start track# 00-76  >00
input start sector# 01-26  >01
   D)ump 1 sector,    C)heck comp.error,    A)uto dump & comp.error check,
   N)ext dump,    X) any dump,    R)eboot CP/M,
input  D,  C,  A,  N,  X,  R,  >C

COMPARE ERROR ON TRACK= 02  SECTOR= 01 -------比較エラーあり.

    +++++ END OF LAST TRACK +++++
    !!!!! COMPARE ERROR EXIST !!!!!

COMPARE ERROR ON TRACK= 02  SECTOR= 01 --------該当セクタを同時ダンプ.
                    ←—エラー箇所.
A:00   00 4D 4F 56 43 50 4D 20 20 43 4F 4D 00 00 00 4C   .MOVCPM  COM...L
B:00   E5 =  =  =  =  =  =  =  =  =  =  =  =  =  =  =   .MOVCPM  COM...L

A:10   02 03 04 05 06 07 08 09 0A 0B 00 00 00 00 00 00   ................
B:10   =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =    ................

A:20   00 50 49 50 20 20 20 20 20 43 4F 4D 00 00 00 3A   .PIP     COM...:
B:20   =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =    .PIP     COM...:

A:30   0C 0D 0E 0F 10 11 12 13 00 00 00 00 00 00 00 00   ................
B:30   =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =    ................

A:40   00 53 55 42 4D 49 54 20 20 43 4F 4D 00 00 00 0A   .SUBMIT  COM....
B:40   =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =    .SUBMIT  COM....

A:50   14 15 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
B:50   =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =    ................

A:60   00 58 53 55 42 20 20 20 20 43 4F 4D 00 00 00 06   .XSUB    COM....
B:60   =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =    .XSUB    COM....

A:70   16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
B:70   =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =    ................
 * input any key  >
```

Figure-3.4.17　ERA コマンド実行前のものと，実行後の２枚のディスケットを比較する．

このＡ：，Ｂ：，２枚のディスケットを比較します．まず〝CMPDISK〟を起動して，ディスクの最初のトラック，セクタから最終まで，ディスケット全面の比較チェックを行いました．結果は Figure-3.4.17 に示されているように，トラック02のセクタ01にのみ，比較エラーが見つかりました．ここがディレクトリ部であることは，「DIR コマンド」の項で解説しています．Figure-3.4.2 の ERA コマンドを実行したことによるディスケット上のデータの変化は，トラック02のセクタ01のみであることが分かります．

次にこのセクタを，Ａ：，Ｂ：同時にダンプしたものが Figure-3.4.17 の続きに示されています．16進ダンプ部のＢ：側の〝＝〃は，Ａ：と等しいバイトを表します．どこのバイトがＡ：，Ｂ：異なっているのでしょう．そう，〝MOVCPM．COM〃の32バイトのディレクトリの最初のバイトが00→E5に変っていますね．このように，ERA コマンドはファイルの実際の内容を消去するのではなく，ディレクトリの最初の１バイトを〝E5H〃に書き替えるだけであることが理解されたと思います．よって一旦削除されたファイルでも，その後に書き込みなどが行われていなければ，ディスク上のこの部分を書き替えることにより〝生き返らせる〃ことも不可能ではないわけです．

しかしこれらの物理的なことを，我々ユーザーが気にする必要はありません．ERA コマンドは，あくまでもファイルの〝削除〃であり，ERA コマンドを実行すると，そのファイルは〝消えてなくなる〃と思って運用すればよいのです．

3.5　SAVE　（メモリ内容ディスク・セーブ・コマンド）

――SAVE コマンド――

コマンド形式・機能

SAVE␣n␣x : filename.ext

ドライブ x ：上に，TPAの始まりからnページ分のメモリ・イメージを，ファイル名 〝filename.ext〟としてセーブする．

注1）TPAの始まりはアドレス100Hから．1ページは256バイト（nは10進数）．

注2）x ：はそれがログイン・ディスクの場合，省略できる．

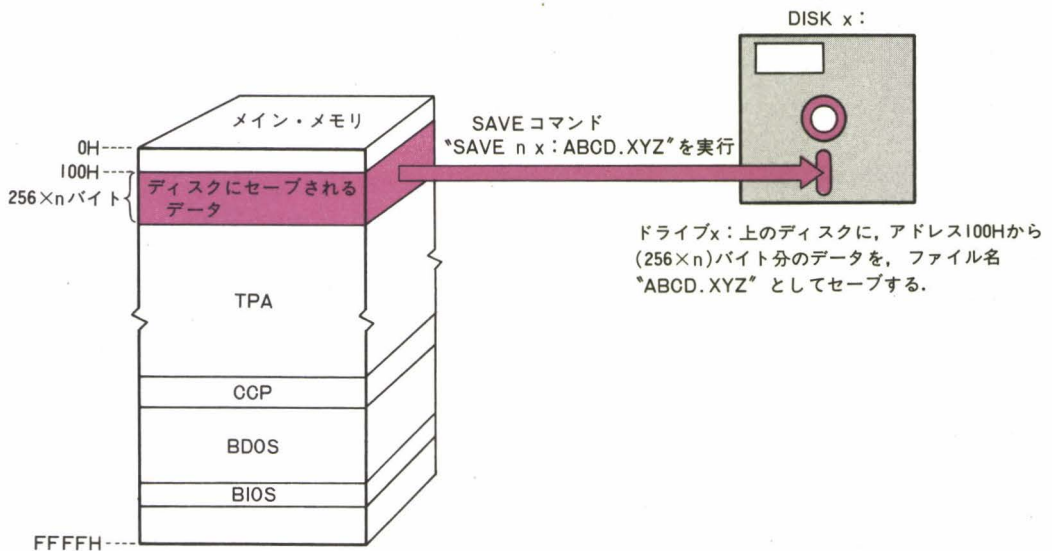



Figure-3.5.0　SAVEコマンドの機能

実習　SAVE␣n␣x:filename.ext

ログイン・ディスク上に TPA のメモリ・イメージをnページ分ファイルする．

現在の TPA のメモリ内容は，何が残っているのか不明なので，まず実行可能な簡単なプログラムをアドレス100Hから書き込み，それをディスクにセーブしてみましょう．書き込むプログラムは「ア

ドレス4000Hから 4FFFH までの 4 Kバイトに，00から始まり，1バイトごとに01，02，03……とインクリメントされるデータを書き込んで行く」という働きをします．このプログラムは僅か17バイトですが，SAVE する時は実習の意味で 2 ページ分ファイルしてみます．あとで確認しやすいよう，プログラムを書き込む前にアドレス100H～4FFH に〝55H〞（ASCII コードのU）をフィルし（満す）しておきます．これらの作業はのちほど解説する DDT を使って行います．

その様子を示します．

```
A>DDT↲ ---- DDTを起動.

DDT VERS 2.2
-F100,4FF,55↲---- アドレス100H～4FFHにデータ55Hをフィル.
-A100↲ ---------- アドレス100Hからライン・アセンブルを開始.
0100  LXI H,4000↲
0103  MVI B,0↲
0105  MOV M,B↲
0106  INX H↲          プログラムを入力して行く.
0107  INR B↲          入力と同時に，対応するオブジェクトが
0108  MOV A,H↲        メモリ上に出来て行く.
0109  CPI 50↲
010B  JNZ 105↲
010E  JMP 0↲
0111  ↲-------- 最後は↲のみで，ライン・アセンブルを終る.

-D100,51F↲----- 結果をダンプして確認.
0100 21 00 40 06 00 70 23 04 7C FE 50 C2 05 01 C3 00 !.@..p#.|.P.....
0110 00 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 .UUUUUUUUUUUUUUU
0120 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
0130 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
0140 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
0150 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
0160 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
0170 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
0180 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
   .
   .
   .
04B0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
04C0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
04D0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
04E0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
04F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 UUUUUUUUUUUUUUUU
0500 C3 15 00 7A E6 38 0F 0F 0F C9 CD 03 03 87 4F 21 ...z.8........O!
0510 42 06 09 4E CD 15 00 23 4E CD 15 00 0E 20 CD 15 B..N...#N.... ..

-^C ------ Ctrl-Cでリブートして CP/Mに戻る.
A>
```

Figure-3.5.1　ディスク上にセーブするデータを作成.

このように目的のプログラムを，アドレス100Hから，DDTの〝A″コマンド（ライン・アセンブラ）で直接書き込みます．

データが出来たら，TPA　の2ページ分（100H～2FFH）をセーブします．このファイルには〝TESTSAVE．COM″という名を付けました．即実行可能なマシン語ファイルなので，ファイル名のエクステンションは〝COM″とします．

```
A>SAVE 2 TESTSAVE.COM ↲
A>
```

Figure-3.5.2
SAVE コマンドの実行．
256×2バイト分をセーブする．

SAVEコマンドの実行が終りました．コマンド・ラインの〝2″は2ページ分を指定する数字です．では，セーブされた〝TESTSAVE．COM″をDIRで確認してみましょう．

```
A>DIR TESTSAVE.COM ↲
A: TESTSAVE COM
A>
```

Figure-3.5.3
結果の確認．

確かにファイルが生成されています．どのようなファイルなのか，このファイルの状態をSTATで調べてみます．（STATコマンドについては第4章「STATコマンド」の項を参照．）

```
A>STAT TESTSAVE.COM ↲

 Recs  Bytes  Ext Acc
    4     1k    1 R/W A:TESTSAVE.COM
Bytes Remaining On A: 94k

A>
```

Figure-3.5.4
セーブされたファイルの状態を確認．

このように〝TESTSAVE．COM″は，ファイル長が1Kバイトであることが分かります．2ページ分，256×2バイトしかセーブしていないのに，1Kバイトのファイルが出来ている理由は，8インチの標準ディスクでは，CP/Mのファイル管理が最小でも1Kバイト単位で行われているからです．

生成されたファイルの内容は，DUMPコマンドでも見ることができます．そのサンプルランを次に

示します（DUMP については，第4章「DUMP コマンド」の項を参照）．

```
A>DUMP TESTSAVE.COM ↲

0000 21 00 40 06 00 70 23 04 7C FE 50 C2 05 01 C3 00
0010 00 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0020 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0030 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0040 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0050 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0060 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0070 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0080 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0090 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
    .
    .
    .
    .
0170 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0180 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
0190 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
01A0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
01B0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
01C0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
01D0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
01E0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55
01F0 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55

A>
```

Figure-3.5.5　DUMP コマンドによるファイルの内容の確認．

このようにマシン語のプログラムと，〝55H〟のコードが2ページ分ファイルされていることが確認できました．

SAVE コマンドとは直接関係ありませんが，COM ファイルとしてセーブされた，この短いプログラムを実行してみましょう．アドレス100Hスタートのプログラムなので，このプログラム名のプライマリ・ネームをキーインし，リターンすることにより自動的に実行されます．

```
A>TESTSAVE ↲

A>
```

Figure-3.5.6
セーブされたプログラムの実行．

このプログラムの最後は〝JMP 0〟のリブート（ウォーム・ブート）になっていますので，実行が終るとこのようにリブートして CP／M にもどります．

一応，実行結果を DDT のダンプ・コマンドで，4000H～4FFFHがどうなっているか見てみましょう．

```
A>DDT
DDT VERS 2.2
-D3FE0,501F
3FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4000 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ................
4010 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ................
4020 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F  !"#$%&'()*+,-./
4030 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
4040 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
4050 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[¥]^_
4060 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 'abcdefghijklmno
4070 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz{|}~.
4080 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F ................
4090 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F ................
40A0 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF ................
40B0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF ................
40C0 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ................
40D0 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF ................
40E0 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF ................
40F0 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF ................
4100 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ................
4110 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F ................
 .
 .
 .
 .
 .
4FB0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF ................
4FC0 C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF ................
4FD0 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF ................
4FE0 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF ................
4FF0 F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF ................
5000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
5010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

-^C
A>
```


Figure-3.5.7　DDT による実行結果の確認.

このように目的通り，4000Hから00，01，02……と書き込まれていることが確認されます．

DDT と併用し，既存ファイルをドライブｘ：上にコピーする．ディスク・ドライブが１台でも，ファイルのコピーが可能．

DDT と SAVE コマンドを使って，PIP コマンドを用いないで，既存ファイルのコピーができます．但し，ファイルが大きくて，TPA にロードしきれないものはダメです．

では，CP／M のアセンブラである〝ASM．COM〞をこの方法でコピーしてみましょう．せっかく DDT を使うのに，ただコピーするだけではつまらないので，アセンブラのエラー・メッセージの一部を，カナで表示するように変更してコピーします．

```
A>STAT ASM.COM ↵

 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASM.COM
Bytes Remaining On A: 105k

A>
```

Figure-3.5.8
ファイル〝ASM．COM〞のファイル長の確認．
Recs の項の「64」をメモしておく．

```
A>ASM XXXX ↵
CP/M ASSEMBLER - VER 2.0
NO SOURCE FILE PRESENT ←このメッセージを変更する．

A>
```

Figure-3.5.9
アセンブラのエラー・メッセージの１つの出力例．この〝NO SOURCE……〞をカナに変更してコピーする．

```
A>DDT ASM.COM ↵ -----DDTを起動して，ASM.COMをTPAにロードする．

DDT VERS 2.2
NEXT  PC
2100 0100

-DFB0,FCF ↵ ------変更しようとするエラー・メッセージの部分のダンプ．
0FB0 20 56 45 52 20 32 2E 30 0D 4E 4F 20 53 4F 55 52  VER 2.0.NO SOUR
0FC0 43 45 20 46 49 4C 45 20 50 52 45 53 45 4E 54 0D CE FILE PRESENT.

-SFB9 ↵  ------Sコマンドでカナ文字のコードに書き替えて行く．
0FB9 4E BF ↵ ソ
0FBA 4F B0 ↵ ー
0FBB 20 BD ↵ ス
0FBC 53 20 ↵ ␣
0FBD 4F CC ↵ フ
0FBE 55 A7 ↵ ァ
0FBF 52 B2 ↵ イ
0FC0 43 D9 ↵ ル
0FC1 45 20 ↵ ␣
0FC2 20 B6 ↵ カ
0FC3 46 DE ↵ ゛
0FC4 49 20 ↵ ␣
```

```
0FC5 4C B1↲ ア
0FC6 45 D8↲ リ
0FC7 20 CF↲ マ
0FC8 50 BE↲ セ
0FC9 52 DD↲ ン
0FCA 45 20↲ ␣
0FCB 53 20↲ ␣
0FCC 45 20↲ ␣
0FCD 4E 20↲ ␣
0FCE 54 20↲ ␣
0FCF 0D . ↲-------ピリオドでSコマンドを終る.

-DFB0,FCF↲ ------変更した部分の確認のダンプ.
0FB0 20 56 45 52 20 32 2E 30 0D BF B0 BD 20 CC A7 B2  VER 2.0.ソース ファイ
0FC0 D9 20 B6 DE 20 B1 D8 CF BE DD 20 20 20 20 20 0D ル ガ アリマセン      .

-^C ------Ctrl-CによりDDTを終了しCP/Mに戻る.
A>
```

Figure-3.5.10　DDT により ASM.COM を TPA にロードし，一部に変更を加える．

```
A>SAVE 32 ASMSAVE.COM ↲
A>
```

Figure-3.5.11
一部変更された TPA の ASM.COM のメモリ・イメージを，先ほどメモした「64」を1／2したページ数分 SAVE する(もし割り切れない場合は切り上げ)．

```
A>STAT ASMSAVE.COM ↲

 Recs  Bytes  Ext Acc
   64     8k    1 R/W A:ASMSAVE.COM
Bytes Remaining On A: 94k

A>
```

Figure-3.5.12
STAT による SAVE されたファイルの確認．オリジナルと同じ容量のファイルがセーブされている．

```
A>ASMSAVE XXXX↲
CP/M ASSEMBLER - VER 2.0
ソース ファイル ガ アリマセン ←――――カナに変更された.

A>
```

Figure-3.5.13
メッセージの一部を変更して，セーブされた新しいアセンブラで，ファイル〝XXXX〟をアセンブルする．エラー・メッセージがカナで出力されていることに注目．

注)　カナ文字を表示するには，もちろんカナ文字が使える CP／M でなくてはなりません．国産のパーソナル・コンピュータ用の CP／M では，ほとんどでカナの使用が可能です．

3.6　USER　(ユーザー・エリア移行コマンド)

——USER　コマンド——

コマンド形式・機能

USER␣n
ユーザー・エリアnに移行する. n＝0～15の10進数.

解説　USER

USER コマンドは，MP／M（マルチ・ユーザーの CP／M）で使用するディスクとのコンパチビリティを持たせるために，CP／M Version 2.0 から追加されました．複数の人が同時に一つの装置を使用する MP／M において，この USER コマンドがなぜ必要なのか，本項により明らかになるでしょう．

CP／M では，大容量のハード・ディスクでも使わない限り，通常は USER コマンドを使う必要性はまったくありません．USER コマンドより，実際のディスケットを何枚か使うことをお勧めします．

さて USER コマンドは，各ドライブ上の1枚のディスケットを仮想の16枚のディスケットに分割し，その16枚に0から15までの番号を付け，コマンド〝USER　n〞を実行することにより，あたかもn番目のディスケットのみを単独で使っているような効果を与えるものです．言い替えれば，物理的に同

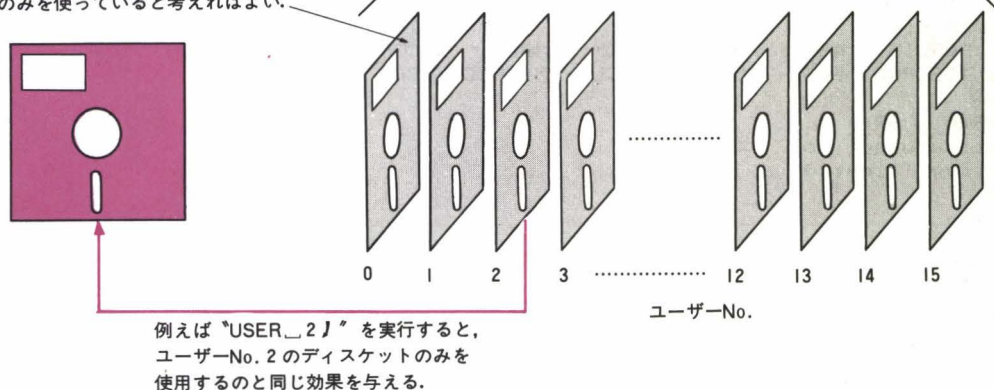



Figure-3.6.0　USERコマンドの機能

一ディスケットであっても，ユーザー・エリアが異なれば別のディスケットと同等なのです．異なるユーザー・エリアのファイルに対しては，ディレクトリすら表示できず，すべてのコマンドが無効です(PIP コマンドの特別な使い方を除く)．要するに，物理的には同一ディスケット上のファイルであっても，ユーザー・エリアが異なれば，それらは存在しないのです．

ユーザー・エリアは，すべてのドライブに対して同時に効果を与えます．ドライブA：がユーザー・エリア5なら，ドライブB：も5になっています．

CP／M が起動した時のデフォルトのユーザー・エリアは0であり，この0エリアはCP／M Version 1.4 のディスケットとコンパチビリティーのある唯一のエリアです． 0以外のユーザー・エリアでは，Version 1.4 のディスケットは読めません．

実習 USER␣n

```
A>DIR ↲ -----ドライブA：のディレクトリを見る.
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM

A>USER 5 ↲ ------ユーザー・エリア5に移行する.

A>DIR ↲ -------再びDIRを実行.
NO FILE -------ファイルがない.
A>DIR B: ↲ -----ドライブB：のディレクトリを見る.
NO FILE -------こちらにも何もない.

A>USER 0 ↲ -----ユーザー・エリア0に帰る.

A>DIR B: ↲ -----B：上のディレクトリを見る. エリア0にはファイルがあった.
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM

A>STAT USR: ↲ -----現在のユーザー・エリアと，ファイルが存在しているユーザー・エリアを表示させる.
Active User : 0 ----現在のエリアは0.
Active Files: 0 ----ファイルの存在するエリアは0のみ.
A>
```

Figure-3.6.1 USER コマンドの実習.

0以外のユーザー・エリアへ，ファイルをコピーする実例が，第4章の「STAT コマンド」の実習10と，PIP コマンドの「[Gn] PIP パラメータ」の項にありますので参照して下さい．また，〝STAT USR：〃の使い方も，「STAT コマンド」の項を参照して下さい．

4章　トランジェント・コマンド徹底実習

4.1　STAT　（ファイルや周辺装置の設定および状況報告プログラム）

——STATistical information program——

コマンド形式・機能

〈ファイルに関するコマンド〉

1　STAT

コールド・スタートやウォーム・スタートの後，アクセスされたことのあるディスクの未使用エリアの容量と，それらのドライブがR／WかR／Oかをレポートする．

2　STAT␣x :

ドライブｘ：の未使用エリアの容量のみレポートする．

3·a　STAT␣x : filename.ext

ドライブｘ：上のファイル〝filename.ext〟の，各種サイズとアトリビュートをレポートする．

3·b　STAT␣x : filename.ext␣$S

3·aと同様であるが，さらにファイルのサイズの表示も行われる．

4·a　STAT␣x : filematch

3·aにファイル・マッチを使用したもの．

4·b　STAT␣x : filematch␣$S

4·aと同様であるが，さらにファイルのサイズの表示も行われる．

5　STAT␣x : filename.ext␣$atr

ドライブｘ：上のファイル〝filename.ext〟に対して，アトリビュート〝atr〟を設定する（CP／M Version 1.4にはこの機能なし）．

6　STAT␣x : filematch␣$atr

5にファイル・マッチを使用したもの．

〈周辺装置に関するコマンド〉

7　STAT␣DEV:

周辺装置の現在の割り付け状況をレポートする．

8　STAT␣VAL:

STATコマンド・ライン・メニューと，ロジカル・デバイス対フィジカル・デバイスの一覧表を表示する（CP／M Version 1.4には，コマンド・メニューはなし）．

9 STAT␣logdev: =phydev:

各周辺装置（フィジカル・デバイス）をロジカル・デバイスに割り付ける.

10 STAT␣USR:

現在のユーザー・エリアとファイルが存在するユーザー・エリアをレポートする（CP／M Version 1.4には，この機能なし).

11 STAT␣x: DSK:

ドライブx：の記憶容量や入力可能ファイル数など，ディスク・ドライブの諸元を表示する（CP／M Version 1.4にはこの機能なし).

12 STAT␣x: =R/O…(この場合のx：は省略できない.)

ドライブx：にリード・オンリーのライト・プロテクトを設定する.

注） x：は，それがログイン・ディスクの場合，省略できる.

実習1 STAT

コールド・スタートやウォーム・スタートの後，アクセスされたことのあるディスクの未使用メモリ・スペースとそれらのドライブがR/Wであるか，R/Oであるかをレポートする.

```
A>STAT ⏎
A: R/W, Space: 106k

A>
```

Figure-4.1.1
ドライブB：には，まだアクセスしたことがないので，ドライブA：に関してのみレポートされている.

```
A>DIR B: ⏎
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
A>
```

Figure-4.1.2 実習のためにDIRを使って，ドライブB：をアクセスしてみる.

```
A>STAT ⏎
A: R/W, Space: 106k
B: R/W, Space: 78k

A>
```

Figure-4.1.3
今回はドライブ A：，B：ともにレポートされた．

このようにCP/Mは，アクセスしたすべてのドライブのディレクトリ情報を，メモリ内に常に保持しています．

実習2 STAT␣x:

ドライブx：に関して，未使用のメモリ・スペースのみレポートします．

```
A>STAT B: ⏎

Bytes Remaining On B: 78k

A>
```

Figure-4.1.4
ドライブB：の未使用メモリ・スペースがレポートされている．
Figure-4.1.3 と比較して下さい．

実習3・a STAT␣x:filename.ext
実習3・b STAT␣x:filename.ext␣$S
実習4・a STAT␣x:filematch
実習4・b STAT␣x:filematch␣$S

ドライブx：上のファイル〝filename. ext〟に関するロジカル・エクステント中のレコード数(Recs)，ファイルの全容量のキロバイト数（Bytes)，16Kバイトのエクステント数（Ext)，ファイルのアクセス・アトリビュート（Acc）などをレポートします．

```
A>STAT B:M80.COM ⏎

 Recs  Bytes  Ext Acc
  137    18k    2 R/W B:M80.COM
Bytes Remaining On B: 78k

A>
```

Figure-4.1.5
ドライブB：上の〝M80. COM〟（マイクロソフト社のマクロアセンブラ）に関してレポートされた．
ドライブB：には Figure-4.1.2 のディスケットが入っている．

また，ファイル名のあとに，〝$S″オプションを付けてコマンドを実行することにより，さらにファイルに割り当てられたレコード数（Size）も表示されます．

```
A>STAT B:M80.COM $S ↲

 Size  Recs  Bytes  Ext Acc
  137   137    18k    2 R/W B:M80.COM
Bytes Remaining On B: 78k

A>
```

Figure-4.1.6　〝Size″ が表示されている．先ほどの Figure-4.1.5 と比較して下さい．通常のファイル（シーケンシャル・ファイル）では 〝Size″ と 〝Recs″ の値は一致する．

次に，ファイル名にファイル・マッチを使ってみましょう．

```
A>STAT B:*.REL ↲

 Recs  Bytes  Ext Acc
  377    48k    3 R/W B:BASLIB.REL
  182    23k    2 R/W B:FORLIB.REL
Bytes Remaining On B: 78k

A>
```

Figure-4.1.7
プライマリ・ファイルネームにファイル・マッチを使い，ドライブB：上のすべての〝REL″ファイルをレポートしている．

ここで，こられのコマンドにより表示される，各項目の用語の説明をしておきましょう．

Size …………シーケンシャル・ファイルの場合は，次の〝Recs″の値と一致し，ファイルに割り当てられたレコード数（1レコードは128バイトで，CP／M上の1セクタに当る）を示します．
ランダム・アクセス・ファイルの場合は，ファイルの最終（End Of File）レコードの番号（位置）を示します．

Recs …………シーケンシャル・ファイルの場合は，そのファイルのロジカル・エクステント（標準ディスクの場合は，16Kバイトのブロックを1ロジカル・エクステントで表す．）によって占められるレコード数を示します．この場合は，〝Size″の値と一致します．
ランダム・アクセス・ファイルの場合は，データが書き込まれていないレコードをも含んだ，ロジカル・エクステントのレコード数を示します．

Bytes …………シーケンシャル・ファイルの場合は，ファイルによって占められる実際のバイト数（ブロック単位であり，標準ディスクの場合は1024バイト＝1Kバイトが最小の単位となる）を示しています．
ランダム・アクセス・ファイルの場合は，実際にデータの書き込みが行われたレコードの合計のバイト数が示されます．ランダム・アクセス・ファイルの場合は，この〝Bytes〟だけが正確なファイルの容量を示す値になります．

Ext …………ロジカル・エクステントの数を示します．8インチの標準ディスケットの場合は，1ロジカル・エクステントにより，16Kバイトのディスク・アロケーション・マップ（1Kバイト・ブロックごとのファイル・データの住所録）を表します．よって，ファイルの容量が16Kバイト以下であれば，〝Ext〟の値は1となり，16Kより大きく，32K以下の場合は2という具合になります．

Acc …………ファイル・アクセスのタイプ（アトリビュート）を示します．〝R/O〟か〝R/W〟かのいずれかを表示します．〝SYS〟アトリビュートは，ファイル名に〝( )〟を付けて表示されます．

実習5 STAT␣x:filename.ext␣$atr
実習6 STAT␣x:filematch␣$atr

ドライブx：上のファイル〝filename．ext〟に対して，アトリビュート〝atr〟($R/O, $R/W, $SYS, $DIR) を設定します．ファイル名には，ファイル・マッチを使うこともできます．

ここで，ファイルのアトリビュート（属性）の種類について解説しましょう．

$R/O …………Read Only File. このアトリビュートが付くと，ファイルは ERA コマンドや REN コマンド，それに ED や同一ファイル名での SAVE や PIP などの実行によるファイルの削除や，一部の変更などに対して，ライト・プロテクトがかかります．

$R/W …………Read/Write File. 通常のファイルが持つアトリビュートであり，自由にファイルの削除や一部の変更ができます．

$SYS …………SYStem File. このファイルは，DIR コマンドによる表示には現れません．また，PIPコマンドによる転送も不可能です．しかし，ERA コマンドや REN コマンドなどは有効です．この〝SYS〟と〝R/O〟の2つのアトリビュートを同

時に付けることが，ファイルを保護するための最高のライト・プロテクトになります．

$DIR …………DIRectory File. DIR コマンドにより表示される通常のファイル．"SYS" アトリビュートの付いたファイルを通常のファイルにもどすときに使います．

これらのアトリビュートの実習の前に，比較の意味で通常のアトリビュートの状態($R/W で $DIR)における各ファイルの DIR コマンドと STAT コマンドによる表示を示しておきます．

```
A>DIR B: ┘
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
A>
```

Figure-4.1.8　通常のアトリビュートでの表示例．
ドライブB：上のすべてのファイルが表示されている．

```
A>STAT B:*.* ┘

 Recs  Bytes  Ext Acc
  227    29k    2 R/W B:BASCOM.COM
  377    48k    3 R/W B:BASLIB.REL
  182    23k    2 R/W B:FORLIB.REL
   59     8k    1 R/W B:L80.COM
  137    18k    2 R/W B:M80.COM
  188    24k    2 R/W B:MBASIC.COM
  102    13k    1 R/W B:TESTPRO.BAS
Bytes Remaining On B: 78k

A>
```

Figure-4.1.9
通常のアトリビュートでの表示例．

では，アトリビュートを付けてみましょう．

```
A>STAT B:*.COM $SYS ┘

BASCOM.COM set to SYS
M80.COM set to SYS
L80.COM set to SYS
MBASIC.COM set to SYS
A>
```

Figure-4.1.10
ドライブB：上のすべての "COM" ファイルに，"SYS"アトリビュートを付ける．アトリビュートが設定されたメッセージが出力されている．

```
A>DIR B:
B: BASLIB   REL : TESTPRO  BAS : FORLIB   REL
A>
```

Figure-4.1.11　DIR コマンドでは，このように〝SYS〞アトリビュートの付いたファイルは表示されない．Figure-4.1.8 と比較して下さい．

```
A>PIP A:=B:M80.COM

NO FILE: =B:M80.COM

A>
```

Figure-4.1.12
PIP コマンドで〝SYS〞アトリビュートの付いたファイルを転送しようとしても，このように〝ファイルなし〞のメッセージが出力され実行できない．

```
A>STAT B:TESTPRO.BAS $R/O

TESTPRO.BAS set to R/O
A>
```

Figure-4.1.13
ドライブ B：上のファイル〝TESTPRO. BAS〞に，〝R／O〞のアトリビュートを付ける．メッセージが出力されてアトリビュートが設定された．

```
A>ERA B:TESTPRO.BAS
Bdos Err On B: File R/O
A>
```

Figure-4.1.14
この〝R／O〞ファイルを ERA コマンドで削除しようとしても，このように削除できない．同様に REN コマンド（リネーム）も実行できない．

```
A>ED B:TESTPRO.BAS

** FILE IS READ/ONLY **
    : *Q

Q-(Y/N)?Y

A>
```

Figure-4.1.15
〝R／O〞ファイルをエディットしようとすると，このように〝リード・オンリー〞である旨のメッセージが出力される．アペンドして，ファイル内容の確認などはできても最後の〝E〞コマンドは実行することはできず，要するにエディットすることは不可能．

```
A>PIP B:TESTPRO.BAS=DUMP.ASM ↲

DESTINATION IS R/O, DELETE (Y/N)?N
**NOT DELETED**

A>
```

Figure-4.1.16
PIP コマンドにより，同一ドライブ上に〝R／O″ファイルと同一ファイル名で，何らかのファイルをコピーしようとすると，このようなメッセージが出力される．すでに存在しているファイルを削除してもよいなら，〝Y″を入力すればコマンドが実行され，〝Y″以外のものを入力すれば何も起らず CP／M にもどる．
同様に SAVE コマンドからも，〝R／O″ファイルはプロテクトされる．

また，〝SYS″ と 〝R／O″ の2つのアトリビュートを同一ファイルに付けることができます．

```
A>STAT B:M80.COM $R/O ↲

M80.COM set to R/O
A>
```

Figure-4.1.17
すでに 〝SYS″ アトリビュートの付いたファイルに，さらに 〝R／O″ を付けた．
この状態が最高のライト・プロテクトになる．

この状態で，ドライブ B：上のファイルの，DIR コマンドと STAT コマンドによる表示を見てみましょう．DIR コマンドによる表示は Figure-4.1.11 と同様です．

```
A>STAT B:*.* ↲

 Recs  Bytes  Ext Acc
  227    29k    2 R/W B:(BASCOM.COM)    ← "SYS"
  377    48k    3 R/W B:BASLIB.REL
  182    23k    2 R/W B:FORLIB.REL
   59     8k    1 R/W B:(L80.COM)       ←
  137    18k    2 R/O B:(M80.COM)       ← "SYS" でかつ "R/O"
  188    24k    2 R/W B:(MBASIC.COM)    ←
  102    13k    1 R/O B:TESTPRO.BAS     "R/O"
Bytes Remaining On B: 78k

A>
```

Figure-4.1.18　このように 〝Acc″ の項と，〝(　)″ で囲まれたファイル名に注目して下さい．初期状態の Figure-4.1.9 とも比較して下さい．

これらのアトリビュートの設定は，ディスク上の各ファイルのディレクトリに直接書き込まれます．よって，リブートやコールド・スタートを行っても消滅することはありません．但し，PIP コマンド

や DDT+SAVE コマンドでファイルをコピーした場合，新しく出来たファイルでは，消滅して通常の〝DIR〞と〝R／O〞ファイルになります．

アトリビュートの変更は，再び STAT コマンドで行います．ここで最初の状態（通常のアトリビュート）にもどす作業を行ったサンプルランを示します．

```
A>STAT B:*.COM $DIR ↵
BASCOM.COM set to DIR
M80.COM set to DIR
L80.COM set to DIR
MBASIC.COM set to DIR

A>STAT B:TESTPRO.BAS $R/W ↵
TESTPRO.BAS set to R/W

A>STAT B:M80.COM $R/W ↵
M80.COM set to R/W

A>
```

Figure-4.1.19
今までに変更したアトリビュートを，もとの状態にもどす．〝DIR〞と〝R／W〞を設定し直す．

```
A>STAT B:*.* ↵

 Recs  Bytes  Ext Acc
  227   29k    2 R/W B:BASCOM.COM
  377   48k    3 R/W B:BASLIB.REL
  182   23k    2 R/W B:FORLIB.REL
   59    8k    1 R/W B:L80.COM
  137   18k    2 R/W B:M80.COM
  188   24k    2 R/W B:MBASIC.COM
  102   13k    1 R/W B:TESTPRO.BAS
Bytes Remaining On B: 78k

A>
```

Figure-4.1.20
このように，もとの何もない通常のファイル状態にもどっている．

実習7　STAT␣DEV:

ロジカル・デバイスに対する現在のフィジカル・デバイスの割り付け状態をレポートします．

```
A>STAT DEV: ↵
CON: is CRT:
RDR: is PTR:
PUN: is PTP:
LST: is LPT:

A>
```

Figure-4.1.21
ロジカル・デバイス〝CON：〞には現在 CRT が，〝RDR：〞には PTR が，〝PUN：〞には PTP が，〝LST：〞には LPT が割り当てられていることが，表示されている．

実習8 STAT␣VAL:

VALidデバイス・アサイン（各ロジカル・デバイスに対して，割り付け可能なフィジカル・デバイス）の状態を表示し，同時にSTATコマンドのコマンド・メニューを表示する．

```
A>STAT VAL: ↲

Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status  : DSK: d:DSK:
User Status  : USR:
Iobyte Assign:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
A>
```

Figure-4.1.22 STATコマンドのコマンド・メニューと，ロジカル・デバイスに対するフィジカル・デバイスの選択表が表示されている．

実習9 STAT␣logdev:=phydev:

1つのロジカル・デバイスに対して，それぞれ4つのフィジカル・デバイスの1つを割り付けます．

```
A>STAT CON:=TTY: ↲

A>STAT PUN:=UP1: ↲

A>
```

Figure-4.1.23
"CON：" に対してはTTY，"PUN：" に対してはUP1を割り付けた．今後は，コンソールとしてTTY：（装置名をTTYと呼ぶだけで，何もテレタイプをつなぐ必要はない）が，パンチャ（これも名称だけで，出力装置の意味）としてUP1：が働くことになる．第1章の1.2「周辺装置の拡張」を参照．

```
A>STAT DEV: ↲
CON: is TTY:
RDR: is PTR:
PUN: is UP1:   ← 変更された.
LST: is LPT:

A>
```

Figure-4.1.24
現在のアサイン状況を調べてみると，このように変更されている．Figure-4.1.21 と比較して下さい．

実習10 STAT␣USR:

現在のユーザー・エリアとファイルが存在しているユーザー・エリアを表示します．ユーザー・エリアについては，第3章の「USER コマンド」を参照．

```
A>STAT USR: ↲

Active User : 0
Active Files: 0
A>
```

Figure-4.1.25
フロッピー・ディスクを使用しているシステムでは，普通はユーザー・エリア0だけを使用するので，このように現在のエリアは0で，かつエリア0のみにファイルが存在していることが表示されている．

しかし，上の例ではおもしろくないので，他のユーザー・エリアにファイルを作ってみましょう．〝PIP．COM〃をユーザー・エリア5と8にコピーします．

```
A>DDT PIP.COM ↲ DDTを起動して，PIP.COMをTPAにロードする.
DDT VERS 2.2
NEXT  PC
1E00 0100
-^C ------------------- CP/Mに戻る.
A>USER 5 ↲               ユーザー・エリア5に移行.
A>SAVE 29 PIP.COM ↲----メモリ上のPIP.COMをディスクにセーブ.
A>USER 8 ↲ -------------ユーザー・エリア8に移行.
A>SAVE 29 PIP.COM ↲----メモリ上のPIP.COMをディスクにセーブ.
A>USER 0 ↲ -------------ユーザー・エリア0に戻る.
A>
  以上で，ユーザー・エリア5と8にPIP.COMがセーブ出来た.
```

Figure-4.1.26　SAVE コマンドによる，各ユーザー・エリアへのファイルのコピー．

これで，ユーザー・エリアの5と8に〝PIP．COM〃がコピーされました．再び，さきほどのコマンドを実行してみましょう．

```
A>STAT USR:↲

Active User : 0
Active Files: 0 5 8
A>
```

Figure-4.1.27
ファイルが存在するユーザー・エリアが表示されている．以前のリストと比較して下さい．

実習11 STAT␣x:DSK:

ディスク・ドライブ x：に関する諸元を表示します．〝x：〃が省略された場合は，コールドまたはウォーム・スタート以後，アクセスされたドライブ全部の諸元が表示されます．

```
A>STAT DSK:↲

    A: Drive Characteristics
 1944: 128 Byte Record Capacity
  243: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   26: Sectors/ Track
    2: Reserved Tracks

    B: Drive Characteristics
 1944: 128 Byte Record Capacity
  243: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   26: Sectors/ Track
    2: Reserved Tracks

A>
```

Figure-4.1.28
〝x：〃を省略したので，アクセスされたことのあるA：，B：両ドライブの諸元が表示されている．

```
A>STAT B:DSK: ↓

    B: Drive Characteristics
 1944: 128 Byte Record Capacity
  243: Kilobyte Drive  Capacity
   64: 32  Byte Directory Entries
   64: Checked  Directory Entries
  128: Records/ Extent
    8: Records/ Block
   26: Sectors/ Track
    2: Reserved Tracks

A>
```

Figure-4.1.29
ドライブを〝B：〟と指定した実行例.
ドライブ B：のみに関して表示されている.

ここで，表示されているディスクの諸元について解説しましょう.

表示される各数値は，ディスクの種類やBIOSの作り方によって異なります．ここで表示されているのは，8インチ標準ディスケットの場合の値です.

1944：128 Byte Record Capacity

1レコード128バイトのレコード（セクタ）の，1ドライブに収容可能な全レコード数．バイト数に換算すれば，128×8＝1Kバイトなので，ディスケット1枚に，1944÷8＝243Kバイト収容可能である．26セクタ×（77本－2本）＝1950であるが，1Kバイト（8セクタ）未満は切り捨てる.

243：Kilobyte Drive Capacity

1ドライブに収容可能な2Kバイト（64×32バイト）のディレクトリ・エリアを含む全バイト数．システム・トラックに使用するバイト数は含まれない．上記の換算と一致するはずである.

64：32 Byte Directory Entries

32バイト単位のディレクトリの入力可能総数を示す．この例では最大64個のファイルを収容できることを示している.

64：Checked Directory Entries

CP／M のファイル管理上，CP／M の内部で参照するためのパラメータとして使用される．フロッピー・ディスク・ドライブのような，メディアを交換できるドライブには，上記の〝32 Byte Directory Entries〟の値と同じ値に設定して，ディスケットを交換した場合のチェックを可能とし，ディスクのディレクトリ部の破壊を防止する．ハード・ディスクの場合は，メディア自体の交換はないので（メディアを簡単に交換可能なものも出現しているが），この値は〝0〟とする．

128：Records／Extent

1エクステントで指定可能なレコード数を示す．この場合，1エクステント（1ディレクトリ）で128レコード（128セクタ）のディスク上のアドレスを指定可能であることを示している．つまり，1エクステントで128×128＝16Kバイトを表すことができるわけである．

8：Records／Block

1ブロック当りのレコード数が8レコード＝8×128バイト＝1Kバイトであることを示している．8インチ標準ディスケットの場合は，ファイルの取り扱いの最小単位が8レコード，つまり1Kバイトであるが，容量の大きなドライブになると，2K，4K，8K……バイトなどに設定される．

26：Sectors／Track

1トラック当りのセクタ数を示す．この場合は26セクタである．

2：Reserved Tracks

CP／M システム用の専用トラック本数を示す．この場合は，2本のトラック（トラック#00と#01）が使用されていることを示している．

実習12 STAT␣x:=R/O

ドライブ x：にリード・オンリーのライト・プロテクトを設定します．

```
A>STAT B:=R/O ┘

A>
```

Figure-4.1.30
ドライブ B：を書き込み禁止ドライブに設定する．

```
A>STAT ┘
A: R/W, Space: 90k
B: R/O, Space: 78k
```

Figure-4.1.31
〝R／O〟の確認．
ドライブ B：が〝R／O〟となっていることがわかる．Figure-4.1.3 と比較して下さい．

```
A>ERA B:L80.COM ┘
Bdos Err On B: R/O
```

Figure-4.1.32
〝R／O〟ドライブの効果の確認．
ドライブ B：上のファイルを削除しようと試みるが，このように実行できない．R／Oのライト・プロテクトは，ドライブ B：全体に対して働いている．実習5．6のように〝ファイル〟に対して働いているのではない点に注意．

この〝R／O〟は，ファイルの〝R／O〟アトリビュートと違って，ディスクに書き込まれるわけではありません．リブートまたはコールド・スタートにより消滅します．ご注意下さい．

4.2　PIP　(周辺装置間のデータ転送プログラム)

——Peripheral Interchange Program——

コマンド形式・機能

1　PIP

PIPプログラムを起動する.

〈ディスク間のコピー〉

2　PIP␣d:=s: {filename.ext / filematch}

ドライブd：上（destinationの意味）に，ドライブs：上（sourceの意味）のファイル“filename.ext”，または“filematch”をコピーする.

3　PIP␣d: newname.ext=s: filename.ext

ドライブd：上に，ドライブs：上のファイル“filename.ext”を，ファイル名を“newname.ext”と変えてコピーする.

〈周辺装置間のデータ転送〉

4　PIP␣CON: =s: filename.ext

コンソール・デバイスのいずれかへ，ドライブs：上のファイル“filename.ext”を出力する.

5　PIP␣PUN: =s: filename.ext

パンチ・デバイスのいずれかへ，ドライブs：上のファイル“filename.ext”を出力する.

6　PIP␣LST: =s: filename.ext

リスト・デバイスのいずれかへ，ドライブs：上のファイル“filename.ext”を出力する.

7　PIP␣d: filename.ext=RDR:

リーダ・デバイスのいずれかからデータを入力し，ドライブd：上にファイル“filename.ext”を作る.

8　PIP␣txdev: =rxdev:

入力デバイスからのデータを，出力デバイスへ送り出す.

9　…=s: filename1.ext, s': filename2.ext, s": filename3.ext,…

2～6のコマンド・ラインで，複数のソース・ファイルを連結して，1つのファイルとしてコピー，または連続して周辺装置に出力することができる.

注1) PIPコマンドには，2つの実行方法がある．その1つはPIPプログラムを形式1により，まず最初に起動し，PIPが起動したことを示す〝*〞プロンプトが出力されたら，その〝*〞に続けて，形式2～9の目的を指示するコマンド・ラインを記述して，実行する方法．この場合，実行が終われば再び〝*〞に戻る．
もう1つは，メイン・コマンドの〝PIP〞に続けて，目的を指示するコマンド・ラインを一度に記述し，PIPプログラムの起動と同時にコマンド・ラインの実行もやってしまう方法．この場合は，実行が終わればCP/Mに戻る．前者の方法は，いくつかの目的の異なるコマンド・ラインを次々と実行する場合に便利であり，後者の方法は，ただ一つのコマンド・ラインを実行する場合に便利である．

注2) ドライブ名d：やs：などは，それがログイン・ディスクの場合，省略できる．

注3) もしIOバイトによるフィジカル・デバイス（TTY:,UR1:, UP1:…)がサポートされていれば，4～9のコマンド形式で，フィジカル・デバイス名を直接使ってもよい．

PIPの実習に入る前に，実習に使用するディスケット上のファイルをDIRコマンドで示しておきます．

```
A>DIR↲
A: MOVCPM   COM : PIP      COM : SUBMIT   COM : XSUB     COM
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM : SYSGEN   COM : DUMP     COM : BIOS     ASM
A: CBIOS    ASM : DEBLOCK  ASM : DISKDEF  LIB : PTBIOS48 ASM
A: PTBOOT48 ASM : PTCPM48  COM : DUMP     ASM
A>
```

Figure-4.2.1・a ドライブA：上のディスケットの内容．

```
A>DIR B:↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM
A>
```

Figure-4.2.1・b ドライブB：上のディスケットの内容．

本書の実習では，周辺装置を介して外部の機器とデータ転送を行う場合，主にもう1台のCP/Mマシンと，互いの〝PUN：〞と〝RDR：〞の各デバイスを介して交信することを例題としています．PUN：やRDR：は具体的には，RS-232Cポートなどであるわけです．その様子をFigure-4.2.2に示します．しかし，周辺装置は何であっても，基本的には全く同じです．パンチャやリーダ，あるいはモデムや

MT カセットなど，どのようなものでも，ここでの実習と同じことが言えるのです．

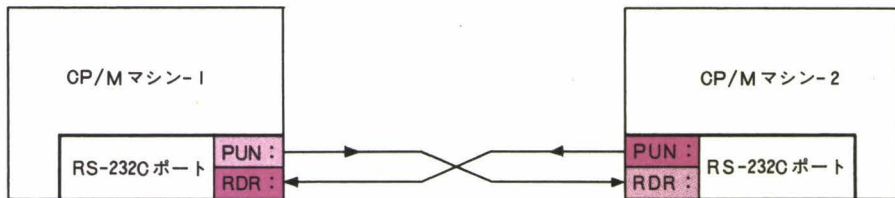



Figure-4.2.2　2台のCP／Mマシン間のデータ転送

実習1　PIP

CP／M のコマンドの中で，PIP と DDT は，それぞれのプログラムのみを，目的を記述したコマンド・ラインを伴わずに，単独で起動することができます．

Figure-4.2.3
PIP が起動し，コマンド・ラインの入力を促すプロント〝*〟が出力された．

実習2　PIP␣d:=s: { filename.ext / filematch }

ドライブ d：上に，ドライブ s：上のファイル〝filename．ext〟をコピーします．この場合，ファイル名にファイル・マッチを使用することもできます．

Figure-4.2.4
コマンド・ラインの〝ED. COM〟は本来〝A：ED. COM〟であるが，現在のログイン・ディスクが A：なので，ドライブ名を省略できる．ドライブ A：上の〝ED. COM〟がドライブ B：上にコピーされた．実行が終ると再び PIP のプロンプトにもどっている．

```
*B:=*.ASM ↲

COPYING -
BIOS.ASM
CBIOS.ASM
DEBLOCK.ASM
PTBIOS48.ASM
PTBOOT48.ASM
DUMP.ASM
*
```

Figure-4.2.5
上記実習のプロンプト〝*″のあとに引き続いて，ファイル・マッチを使った例．ドライブA：上のエクステンションが〝ASM″であるすべてのファイルが，ドライブB：上にコピーされた．実行が終ると再びPIPのプロンプトにもどっている．

ここで実習した，Figure-4.2.3〜5 の PIP コマンドの使い方には，特記しなければならないもう一つの使い方があります．それは「ソース・ディスケット（この場合ドライブA：）を交替しながら，デスティネーション・ディスク上（この場合ドライブB：）にファイルを次々とコピーできる．」ということです．このことは，デスティネーション・ディスケットを任意のドライブに挿入して，PIPを起動しておけば，別々のソース・ディスケットから，必要なファイルだけをコピーすることができるということであり，この手法は，日常よく使われます．試しに，ドライブA：のソース・ディスケットを，別のディスケットと入れ替えて，その中のファイルをドライブB：上へコピーしてみて下さい．

特記事項をもう1つ．PIP を使って，ディスク上のすべてのファイルを，〝*　.*″などで，別の空のディスクにコピーすると，ファイルがきれいに整理されます．エディタや，ERA, SAVE, PIP コマンドなどをたくさん使って作業したディスケットは，適当な時に，ファイルのオール・コピーを行うと，バラバラに散在していた1つのファイルのデータが，一カ所に連続したファイルとなってコピーされますので，ディスクのリード／ライトがずっと速くなります．

さて，Figure-4.2.5 の後から実習を続けます．

```
*A:=B:TESTPRO.BAS ↲
* ↲

A>
```

Figure-4.2.6
今回は，コピーの方向が逆になっている．ドライブB：上のファイル〝TESTPRO. BAS″が，ドライブA：上にコピーされた．実習が終ると再びPIPのプロンプトが出力される．ここではPIPをすべて終了し，CP／Mにもどすために〝↲″をキーインしている．CP／Mにもどったことを示す〝A>″に注目．

Figure-4.2.4〜6 の結果を DIR コマンドで確認してみましょう．

```
A>DIR B:↲
B: BASLIB   REL : BASCOM   COM : M80      COM : L80      COM
B: TESTPRO  BAS : FORLIB   REL : MBASIC   COM : ED       COM
B: BIOS     ASM : CBIOS    ASM : DEBLOCK  ASM : PTBIOS48 ASM
B: PTBOOT48 ASM : DUMP     ASM
A>
```

Figure-4.2.7　Figure-4.2.1-a と比較して下さい．ドライブ B：上には，〝ED. COM〞以後のファイルが新たにコピーされている．

```
A>DIR TESTPRO.BAS↲
A: TESTPRO  BAS
A>
```

Figure-4.2.8
ドライブ A：上には〝TESTPRO. BAS〞がコピーされている．

PIP コマンドを実行するには，二通りの方法があることは前述しましたが，例えば Figure-4.2.4 は次のようになります．

```
A>PIP↲
*B:=ED.COM↲
*
```

Figure-4.2.9・a
実行が終ると再び PIP のプロンプト〝*〞にもどり，次の PIP 操作が可能となる．

‖　　結果は同じ．

```
A>PIP B:=ED.COM↲

A>
```

Figure-4.2.9・b
実行が終ると PIP を終了し，CP／M にもどってしまう．

実習3 PIP␣d:newname.ext=s:filename.ext

ドライブs：上のファイル〝filename．ext″を，ドライブd：上にファイル名を〝newname．ext″と変更してコピーします．

```
A>PIP A:MACRO80.COM=B:M80.COM ↵

A>
```

Figure-4.2.10
ドライブB：上のファイル〝M80．COM″を，ドライブA：上にファイル名を〝MACRO80．COM″としてコピーする．

```
A>DIR MACRO80.COM ↵
A: MACRO80  COM
A>
```

Figure-4.2.11
結果の確認．〝M80．COM″が〝MACRO80．COM″としてコピーされている．

実習4 PIP␣CON:=s:filename.ext

現在のコンソール・デバイスとして割り当てられている装置(周辺装置のアサインについては,「STATコマンド」を参照)に，ドライブs：上のファイル〝filename．ext″を出力します．

```
A>PIP CON:=B:TESTPRO.BAS ↵

10 '===========================================================
20 '
30 '         ITI  CS  UCS  FOR MOUSE
40 '
50 '===========================================================
60 '
70 PRINT "===== DISCRIMINATED AVOIDANCE-SAME PROGRAM WITH 4 BOXES =====
80 PRINT
90 '
100 '***** INITIAL SET *****
110 '
120 DEFINT A-Z     'DEFAIN ALL VALIABLES TO INTEGER
130 '
140 'DISK FILE NAME DEFINE
```

```
150 INPUT "FILE NAME";FILENAMEIN$
160 INPUT "DATE";DATEIN$
170 '
180 'DISK FILE OPEN
190 OPEN "R",#1,FILENAMEIN$
200 '
210 '--- DATA STORE AREA DIM DEFINE ---
220 'DATA BYTE BIT ASSIGNMENT
230 'BIT7,BIT6 ---- 0 0 = NO LEVER RESPONSE
240 '     "     ---- 0 1 = RESPONSE AT CS
250 '     "     ---- 1 0 = RESPONSE AT UCS
260 'BIT5,4,3,2,1,0 ---- LEVER RESPONSE COUNTER AT ITI
   .
   .
   .
   .
   .
   . 実行中に何らかのキーインを行うとブレークがかかり，処理を中止してCP/Mに戻る.

ABORTED: B:TESTPRO.BAS ←――― ブレークのメッセージ.

A>
```

Figure-4.2.12 現在の"CON：" (コンソール)デバイスは，初期状態のままなので，通常その出力はスクリーンに割り付けられている．よって，ドライブB：上のファイル"TESTPRO. BAS"がスクリーン上に転送されることになる．このコマンド・ライン は結果としてTYPEコマンドを使ったのと同じである．

実習5 PIP␣PUN:=s:filename.ext

現在のパンチ・デバイスとしてアサインされている装置に，ドライブs：上のファイル"filename．ext" を出力します．

```
A>PIP PUN:=DUMP.ASM ↲

A>
```

Figure-4.2.13
現在の"PUN：" (パンチ)デバイスにアサインされている装置の1つに，ドライブA：上のファイル"DUMP. ASM" を出力する．この場合も実行中に何らかのキーインによるブレーク機能が働く．

実習6 PIP␣LST:=s:filename.ext

現在のリスト・デバイスとしてアサインされている装置に，ドライブ s：上のファイル "filename.ext" を出力します．

```
A>PIP LST:=DUMP.ASM⏎

;FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;COPYRIGHT (C) 1975, 1976, 1977, 1978
;DIGITAL RESEARCH
;BOX 579, PACIFIC GROVE
;CALIFORNIA, 93950
;
ORG100H
BDOSEQU0005H;DOS ENTRY POINT
CONSEQU1;READ CONSOLE
TYPEFEQU2;TYPE FUNCTION
PRINTFEQU9;BUFFER PRINT ENTRY
BRKFEQU11;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENFEQU15;FILE OPEN
READFEQU20;READ FUNCTION
;
FCBEQU5CH;FILE CONTROL BLOCK ADDRESS
BUFFEQU80H;INPUT DISK BUFFER ADDRESS
;
;NON GRAPHIC CHARACTERS
CREQU0DH;CARRIAGE RETURN
LFEQU0AH;LINE FEED
        .
        .
        .
        .
        .  タブ機能が働いてないことに注目.
        .
```

Figure-4.2.14 現在の"LST："(リスト)デバイスは，初期状態のままで，通常は"LPT：" (プリンタ)にアサインされている．このようにプリンタにドライブ A：上のファイル"DUMP. ASM"が出力されている．タブの機能が働いてないことに注目．これも同様にキーインによるブレークが働く．

実習7　PIP␣d:filename.ext=RDR:

現在のリーダ・デバイスとしてアサインされている装置からデータを入力し，そのデータをドライブ d：上にファイル "filename．ext" としてセーブします．

外部機器からの受信

```
A>PIP RXDUMP.ASM=RDR:[E]↲

;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
      .
      .
      .
;       FIXED MESSAGE AREA
SIGNON: DB      'FILE DUMP VERSION 1.4$'
OPNMSG: DB      CR,LF,'NO INPUT FILE PRESENT ON DISK$'

;       VARIABLE AREA
IBP:    DS      2       ;INPUT BUFFER POINTER
OLDSP:  DS      2       ;ENTRY SP VALUE FROM CCP
;
;       STACK AREA
        DS      64      ;RESERVE 32 LEVEL STACK
STKTOP:
;
        END

A>
```

Figure-4.2.15　現在の "RDR：" (リーダ) デバイスにアサインされている装置からデータを入力し，ドライブ A：上にファイル "RXDUMP. ASM" としてセーブする．入力データは，通常紙テープ・リーダやモデム，それに種々の通信ポートから入力されるが，ここでは，RS-232C ポートに入力される，別の CP/M マシンから送り出された DUMP コマンドのアセンブリ・ソース・ファイルである．
コマンド・ラインの最後の "[E]" は入力データをスクリーン上に表示するためのオプション・コマンドであり，特に付ける必要はない．ファイルの最後を受信すると，自動的にディスクへのセーブが行われる．後述「PIP パラメータ」を参照．

```
A>DIR RXDUMP.ASM ↲
A: RXDUMP    ASM
A>
```

Figure-4.2.16
セーブされたファイルの確認．
このようにファイル〝RXDUMP. ASM〞が作られている．内容は送り出し側のファイル〝DUMP. ASM〞と一致する．

参考までに，送り出し側のCP／Mマシンに与えた〝DUMP. ASM〞を送信するためのコマンドを示しておきます．

外部へ送信

```
A>PIP PUN:=DUMP.ASM,EOF: ↲

A>
```

Figure-4.2.17
前述のFig-4.2.13と同様であるが，最後に特別な装置名の〝EOF：〞（後述）を付ける．この〝EOF：〞を送り出すことにより，受信側はファイルの最後を検知し，自動的にディスクにセーブする．

実習8　PIP␣txdev:=rxdev:

入力デバイスから入力したデータを，PIPを通して（ということは，後述のPIPパラメータによる処理が可能ということ），出力デバイスに出力します．

```
A>PIP PUN:=CON: ↲

ABCDEFGHIJKLMNopqrstuvwxyz
1234567890,.!"#$%&'()=*-+@
TERMINAL MODE by PIP COMMAND
^Z ……Ctrl-ZによりPIPを終了する.
A>
```

Figure-4.2.18
CP／Mマシンをキーボードとして使用する一例．キーインされたものは，〝PUN：〞デバイスから出力されると同時に，スクリーンにも表示される．Ctrl-Zのキーインにより，CP／Mにもどる．Ctrl-Zにより，CP／Mにもどると都合が悪い場合には，コマンド・ラインの最後に後述のPIPパラメータ〝[O]〞を付けて実行しておけばよい．

実習9 …s:filename1.ext, s':filename2.ext,s":filename3.ext…

ファイルを転送するPIP操作の場合，任意のドライブの任意のファイルを，複数個連続して転送することができます．つまり別々のファイルを接続して，1本のファイルとして転送することが可能です．

```
A>PIP B:NEW.TXT=DUMP.ASM,B:BIOS.ASM,DEBLOCK.ASM ↲

A>
```

Figure-4.2.19　ドライブA：上のファイル"DUMP．ASM"とドライブB：上のファイル"BIOS．ASM"とドライブA：上のファイル"DEBLOCK．ASM"を1本のファイル"NEW．TXT"として，ドライブB：上にコピーする．

```
A>TYPE B:NEW.TXT ↲

;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
;
    .
    .
    .
    .
    .
        MVI     C,READF
        CALL    BDOS
        POP B! POP D! POP H
        RET
;
;       FIXED MESSAGE AREA
SIGNON: DB      'FILE DUMP VERSION 1.4$'
```

Figure-4.2.20　その1

```
OPNMSG: DB      CR,LF,'NO INPUT FILE PRESENT ON DISK$'

;       VARIABLE AREA
IBP:    DS      2       ;INPUT BUFFER POINTER
OLDSP:  DS      2       ;ENTRY SP VALUE FROM CCP
;
;       STACK AREA
        DS      64      ;RESERVE 32 LEVEL STACK
STKTOP:
;
        END

;       MDS-800 I/O Drivers for CP/M 2.2
;       (four drive single density version)
;
;       Version 2.2 February, 1980
;
vers    equ     22      ;version 2.2
;
;       Copyright (c) 1980
;       Digital Research
;       Box 579, Pacific Grove
;       California, 93950
;
;
true    equ     0ffffh  ;value of "true"
false   equ     not true        ;"false"
test    equ     false   ;true if test bios
    .
    .
    .
    .
    .
                        ;       10 if drive 2,3
iopb:   ;io parameter block
        db      80h     ;normal i/o operation
iof:    db      readf   ;io function, initial read
ion:    db      1       ;number of sectors to read
iot:    db      offset  ;track number
ios:    db      1       ;sector number
iod:    dw      buff    ;io address
;
;
;       define ram areas for bdos operation
        endef
        end

;*****************************************************
;*                                                   *
;*      Sector Deblocking Algorithms for CP/M 2.0    *
;*                                                   *
;*****************************************************
;
;       utility macro to compute sector mask
smask   macro   hblk
;;      compute log2(hblk), return @x as result
```

接続箇所（分かり易くするために，ライン・スペースを空けてあるが，実際はない）.

接続箇所（前に同じ）.

Figure-4.2.20 その2

```
;;      (2 ** @x = hblk on return)
@y      set     hblk
@x      set     0
;;      count right shifts of @y until = 1
        rept    8
        if      @y = 1
        exitm
        endif
;;      @y is not 1, shift right one position
    .
    .
    .
    .
    .
unacnt: ds      1                       ;unalloc rec cnt
unadsk: ds      1                       ;last unalloc disk
unatrk: ds      2                       ;last unalloc track
unasec: ds      1                       ;last unalloc sector
;
erflag: ds      1                       ;error reporting
rsflag: ds      1                       ;read sector flag
readop: ds      1                       ;1 if read operation
wrtype: ds      1                       ;write operation type
dmaadr: ds      2                       ;last dma address
hstbuf: ds      hstsiz                  ;host buffer
;
;*****************************************************
;*                                                   *
;*      The ENDEF macro invocation goes here.        *
;*                                                   *
;*****************************************************
        end

A>
```

Figure-4.2.20　新しく作られたドライブB：上のファイル〝NEW．TXT〟を TYPE コマンドで確認する．このように別々のファイルであったものが，1本になっていることが確認される．

PIPパラメータ

PIPには，PIPパラメータと呼ばれるオプション・コマンドがあり，その種類を次の表に示します．今まで解説してきたPIPコマンド・ラインの最後に，このPIPパラメータを付けて実行することにより，各種の処理を行いながらデータの転送ができます．

PIPパラメータの機能

[B] (Block)

ブロック・モード転送を行う．〝X-off〟キャラクタであるCtrl-Sを受信するまでは，受信データをバッファにロードして行き，Ctrl-Sを受信すると，バッファにロードされたデータをディスクにセーブし，バッファを空にして，再び受信データのバッファへのロードを開始する．

[Dn] (Delete)

ライン文字数の制限転送．アスキー・ファイルやキャラクタ入力を転送する際，各ラインの頭からn文字を越えた文字を削除する．

[E] (Echo)

装置間で転送されているデータを，コンソールにも同時に出力する．

[F] (Form feed)

転送データからフォーム・フィード・キャラクタ（0CH）を削除する．

[Gn] (Get)

現在のユーザー・エリアにおいて，他のユーザー・エリアnから，PIPによる転送を可能にする．

[H] (Hex format)

インテルHEX形式のデータ転送の際，HEX形式としてのエラーのチェックを行う．

[I] (Ignores null)

上記［H］の機能に，ヌルレコード（：00）を無視する機能を追加したもの．

[L] (Lower case)

すべての大文字を小文字に変換して転送する．

[N] (line Numbers)

転送データにラインNo.を付ける．ラインNo.の数字の前置きの〝0〟を表示させるには［N 2］パラメータを用いる．

[O] (Object files)

転送の際〝EOF：〟(End Of File) キャラクタの〝1AH (Ctrl-Z) によるターミネート処理を

無視する．アスキー・ファイルでないファイルの転送も可能となる．

[Pn] (Page ejects)

nラインごとにページ送りをする．nが1または指定されない時は，60ラインが初期値として指定される．

[Q文字列^Z] (Quit)

転送データをサーチし〝文字列〟が来ると，転送を終了する．〝文字列〟自身も転送される．

[S文字列^Z] (Start)

転送されるデータから〝文字列〟をサーチし見つかれば〝文字列〟自身も含め，以後のデータの転送を開始する．

[R] (Read)

〝SYS〟アトリビュートの付いたファイルの転送を可能にする．同時に [W] パラメータのセットも行われる．

[Tn] (Tab)

転送データ中のタブ・スペースをnカラムに設定する．

[U] (Upper case)

すべての小文字を大文字に変換して転送する．

[V] (Verify)

ディスク間のファイルのコピーの際，リード・アフタ・ライト（書き込み後，即読み出して比較）の確認を行い，コピー・データを保障する．

[W] (Write in R/O)

デスティネーションの〝R/O〟アトリビュートを無視して転送を行う．

[Z] (Zeros parity)

入力デバイスから，データの受信時に受信データのパリティ・ビットを0にする．

実習10 [B]パラメータ

入力装置からデータを入力して，ディスクにセーブする場合，送り出すデータを適当な長さのブロックに区切ることにより，ブロック単位で，バッファ・メモリへの受信⇨ディスクへのセーブ⇨次のブロックのバッファ・メモリへの受信⇨ディスクへのセーブ⇨……をくり返し，カセット・テープや紙テープなどの，シーケンシャルな出力装置からのデータを，ブロック・バッファリングにより，ディスクへのセーブを可能にします．但しアスキー・データが対象となります．

データの送り側が，各ブロックの終りに〝X-off キャラクタ〟である Ctrl-Sを送信することにより，

受信側は，今までに受信されたバッファ上のブロックをディスクにセーブします．すべてのブロックの送り出しが終れば，通常の〝End Of File〃キャラクタのCtrl-Zを送信することにより，最後のセーブを行い，セーブしたファイルをクローズさせファイルが出来上ります．

実習例として，リーダ・デバイス〝RDR：〃からデータを入力し，〝TEST／B．TXT〃というファイル名でディスクにセーブしてみましょう．送り出し側には便宜上CP／Mマシンを使い，そのキーボード入力をパンチ・デバイス〝PUN：〃から出力させます．この場合，受信側の〝RDR：〃と送信側の〝PUN：〃はRS-232Cポートなどで接続されていると考えて下さい．

```
A>PIP TEST/B.TXT=RDR:[B] ↲

A>
```

Figure-4.2.21
受信側でのコマンド・ライン．
パラメータ[B]を付けて，リターンしておけば受信状態となる．〝EOF：〃(Ctrl-Z) を受信すると，PIP を終了し CP/M にもどる．

```
A>PIP PUN:=CON:,EOF: ↲          LF=ラインフィード（LFキーのないキーボードは，Ctrl-Jで代用できる）

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ↲ LF
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB ↲ LF
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC ↲ LF
^S （実際はこのようなライン・スペースは空かない）
1111111111111111111111111111111111111111111 ↲ LF
22222222222222222222222222222222222222222222 ↲ LF
333333333333333333333333333333333333333333333 ↲ LF      キーインして行く．
^S （同上，分かり易くするために空けてある）
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ↲ LF
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy ↲ LF
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz ↲ LF
END OF LIST. ↲
^Z --------- 最後はCtrl-Zで終る．
A>
```

Figure-4.2.22　送信側でのコマンド・ライン．
キーインプットが PUN：から出力される．最後に〝EOF：〃を送信するために，後述の PIP キーワード〝EOF：〃を付ける．キーインされた文字は同時にスクリーンにも表示される．

このように，キー入力した各ブロックの終りにはCtrl-Sをキーインして行き，最後にCtrl-Zをキーインすることにより，受信データがディスクにセーブされ，送信側も受信側もPIPを終了しCP／Mにもどります．

セーブされたファイル "TEST/B. TXT" を TYPE コマンドで確認してみましょう.

```
A>TYPE TEST/B.TXT ↲

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
1111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222
333333333333333333333333333333333333333333333333
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
END OF LIST.

A>
```

Figure-4.2.23　セーブされたファイルの内容の確認.

データ送り出し用のＣＰ／Ｍマシンが用意できない場合は，１台でも次のようなコマンド・ラインにより，[B] パラメータを実習することができます.

```
A>PIP TEST/B.TXT=CON:[B],EOF:↲
        •
        •
 時々Ctrl-S を混えながら適当にキーインして行く.
        •
        •
^Z  --------(最後はCtrl-Zで終る)
A>
```

Figure-4.2.24　１台の CP/M マシンで実習する場合，自分のマシンのキー入力が，直接バッファへの入力となる.

実習11　[Dn]パラメータ

アスキー・ファイルや入力装置からのアスキー・データを転送する際に，各ラインの文字数を最大ｎ文字に制限します.

```
A>PIP CON:=TEST/B.TXT[D20]

AAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBB
CCCCCCCCCCCCCCCCCCCC
11111111111111111111
22222222222222222222
33333333333333333333
XXXXXXXXXXXXXXXXXXXX
YYYYYYYYYYYYYYYYYYYY
ZZZZZZZZZZZZZZZZZZZZ
END OF LIST.

A>
```

Figure-4.2.25
先程のファイル〝TEST／B．TXT〞を１ライン20文字に制限して，コンソールに出力する．

```
A>PIP CON:=CON:[D5]

1122334455567890  LF

AABBCCDDEEFGHIJKLMN  LF

  LF=ラインフィード（LFキーのないキーボードは,
A>                         Ctrl-Jで代用できる)
```

Figure-4.2.26
キー入力を自分のコンソールに１ライン５文字に制限して出力する．最初の５文字が，二重になっていることに注目．コンソール入力（キー入力）は自分自身にコンソールへの表示機能があるために，このように５文字だけが二重となり，［Ｄ５］による効果が確認できる．

実習12　[E]パラメータ

装置間で転送されているデータを，同時にコンソールへも出力します．アスキー･データであれば，スクリーン上で内容の確認ができるわけです．

```
A>PIP B:=DUMP.ASM[E]

;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
```

```
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
;
FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS

        .
        .
        .
        .
        .

;       FIXED MESSAGE AREA
SIGNON: DB      'FILE DUMP VERSION 1.4$'
OPNMSG: DB      CR,LF,'NO INPUT FILE PRESENT ON DISK$'

;       VARIABLE AREA
IBP:    DS      2       ;INPUT BUFFER POINTER
OLDSP:  DS      2       ;ENTRY SP VALUE FROM CCP
;
;       STACK AREA
        DS      64      ;RESERVE 32 LEVEL STACK
STKTOP:
;
        END

A>
```

Figure-4.2.27 ドライブA：上のファイル"DUMP．ASM"をドライブB：上へコピーするコマンドに，［E］を付けると，このように転送内容が同時にスクリーン上で確認できる．

実習13 ［F］パラメータ

転送データの中から，プリンタ用紙のフォーム・フィードや，スクリーンのオール・クリアに使われる "Form-Feed" キャラクタの "0CH" を取り除きます．

```
A>PIP LST:=DUMP.PRN[T8]↓
```

Figure-4.2.28
［F］パラメータを付けずに，MACRO-80でアセンブルしたDUMPプログラムのプリント・ファイルをプリンタへ出力する．［T 8］は後述のタブを8文字に設定するパラメータ．

```
A>PIP LST:=DUMP.PRN[FT8]
```

Figure-4.2.29
今度は，先の例に[F]パラメータを付けて実行する．

```
A>PIP LST:=DUMP.PRN[FT8]
        MACRO-80 3.33   12-Sep-79       PAGE    1

                                ;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HE
                                ;
                                ;       COPYRIGHT (C) 1975, 1976, 1977, 1978
                                ;       DIGITAL RESEARCH
                                ;       BOX 579, PACIFIC GROVE
                                ;       CALIFORNIA, 93950
                                ;
                                        ORG     100H
 0005                           BDOS    EQU     0005H   ;DOS ENTRY POINT
 0001                           CONS    EQU     1       ;READ CONSOLE
 0002                           TYPEF   EQU     2       ;TYPE FUNCTION
 0009                           PRINTF  EQU     9       ;BUFFER PRINT ENTRY
 000B                           BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY
 000F                           OPENF   EQU     15      ;FILE OPEN
 0014                           READF   EQU     20      ;READ FUNCTION
                                ;
 005C                           FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
 0080                           BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
                                ;
                                ;       NON GRAPHIC CHARACTERS
 000D                           CR      EQU     0DH     ;CARRIAGE RETURN
 000A                           LF      EQU     0AH     ;LINE FEED
                                ;
                                ;       FILE CONTROL BLOCK DEFINITIONS
 005C                           FCBDN   EQU     FCB+0   ;DISK NAME
 005D                           FCBFN   EQU     FCB+1   ;FILE NAME
 0065                           FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
 0068                           FCBRL   EQU     FCB+12  ;FILE'S CURRENT REEL NUMBER
 006B                           FCBRC   EQU     FCB+15  ;FILE'S RECORD COUNT (0 TO 128)
 007C                           FCBCR   EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0 TO 127
 007D                           FCBLN   EQU     FCB+33  ;FCB LENGTH
                                ;
                                ;       SET UP STACK
 0100'  21 0000                         LXI     H,0
 0103'  39                              DAD     SP
                                ;       ENTRY STACK POINTER IN HL FROM THE CCP
 0104'  22 0209'                        SHLD    OLDSP
                                ;       SET SP TO LOCAL STACK AREA (RESTORED AT FINIS)
 0107'  31 024B'                        LXI     SP,STKTOP
                                ;       READ AND PRINT SUCCESSIVE BUFFERS
 010A'  CD 01B9'                        CALL    SETUP   ;SET UP INPUT FILE
 010D'  FE FF                           CPI     255     ;255 IF FILE NOT PRESENT
 010F'  C2 011B'                        JNZ     OPENOK  ;SKIP IF OPEN IS OK
                                ;
                                ;       FILE NOT THERE, GIVE ERROR MESSAGE AND RETURN
 0112'  11 01E7'                        LXI     D,OPNMSG
        MACRO-80 3.33   12-Sep-79       PAGE    1-1

 0115'  CD 0194'                        CALL    ERR
 0118'  C3 0151'                        JMP     FINIS   ;TO RETURN
                                ;
                                OPENOK: ;OPEN OPERATION OK, SET BUFFER INDEX TO END
 011B'  3E 80                           MVI     A,80H
 011D'  32 0207'                        STA     IBP     ;SET BUFFER POINTER TO 80H
                                ;       HL CONTAINS NEXT ADDRESS TO PRINT
 0120'  21 0000                         LXI     H,0     ;START WITH 0000
                                ;
                                GLOOP:
 0123'  E5                              PUSH    H       ;SAVE LINE POSITION
 0124'  CD 019A'                        CALL    GNB
 0127'  E1                              POP     H       ;RECALL LINE POSITION
 0128'  DA 0151'                        JC      FINIS   ;CARRY SET BY GNB IF END FILE
 012B'  47                              MOV     B,A
                                ;       PRINT HEX VALUES
                                ;       CHECK FOR LINE FOLD
 012C'  7D                              MOV     A,L
 012D'  E6 0F                           ANI     0FH     ;CHECK LOW 4 BITS
 012F'  C2 0144'                        JNZ     NONUM
                                ;       PRINT LINE NUMBER
 0132'  CD 016A'                        CALL    CRLF
                                ;
                                ;       CHECK FOR BREAK KEY
 0135'  CD 0159'                        CALL    BREAK
                                ;       ACCUM LSB = 1 IF CHARACTER READY
 0138'  0F                              RRC             ;INTO CARRY
 0139'  DA 0151'                        JC      FINIS   ;DON'T PRINT ANY MORE
                                ;
 013C'  7C                              MOV     A,H
 013D'  CD 0187'                        CALL    PHEX
 0140'  7D                              MOV     A,L
 0141'  CD 0187'                        CALL    PHEX
                                NONUM:
 0144'  23                              INX     H       ;TO NEXT LINE NUMBER
 0145'  3E 20                           MVI     A,' '
 0147'  CD 0161'                        CALL    PCHAR
 014A'  78                              MOV     A,B
 014B'  CD 0187'                        CALL    PHEX
 014E'  C3 0123'                        JMP     GLOOP
                                ;
                                FINIS:
                                ;       END OF DUMP, RETURN TO CCP
                                ;       (NOTE THAT A JMP TO 0000H REBOOTS)
 0151'  CD 016A'                        CALL    CRLF
 0154'  2A 0209'                        LHLD    OLDSP
        MACRO-80 3.33   12-Sep-79       PAGE    1-2

 0157'  F9                              SPHL
                                ;       ST
 0158'  C9
                                ;
                                ;

0 0159'  E5
 015A'  0E 0B
 015C'  CD 0005
0 015F'  C1
 0160'  C9
```

```
        MACRO-80 3.33   12-Sep-79       PAGE    1

                                ;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HE)
                                ;
                                ;       COPYRIGHT (C) 1975, 1976, 1977, 1978
                                ;       DIGITAL RESEARCH
                                ;       BOX 579, PACIFIC GROVE
                                ;       CALIFORNIA, 93950
                                ;
                                        ORG     100H
 0005                           BDOS    EQU     0005H   ;DOS ENTRY POINT
 0001                           CONS    EQU     1       ;READ CONSOLE
 0002                           TYPEF   EQU     2       ;TYPE FUNCTION
 0009                           PRINTF  EQU     9       ;BUFFER PRINT ENTRY
 000B                           BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
 000F                           OPENF   EQU     15      ;FILE OPEN
 0014                           READF   EQU     20      ;READ FUNCTION
                                ;
 005C                           FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
 0080                           BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
                                ;
                                ;       NON GRAPHIC CHARACTERS
 000D                           CR      EQU     0DH     ;CARRIAGE RETURN
 000A                           LF      EQU     0AH     ;LINE FEED
                                ;
                                ;       FILE CONTROL BLOCK DEFINITIONS
 005C                           FCBDN   EQU     FCB+0   ;DISK NAME
 005D                           FCBFN   EQU     FCB+1   ;FILE NAME
 0065                           FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
 0068                           FCBRL   EQU     FCB+12  ;FILE'S CURRENT REEL NUMBER
 006B                           FCBRC   EQU     FCB+15  ;FILE'S RECORD COUNT (0 TO 128)
 007C                           FCBCR   EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0 TO 127)
 007D                           FCBLN   EQU     FCB+33  ;FCB LENGTH
                                ;
                                ;       SET UP STACK
 0100'  21 0000                         LXI     H,0
 0103'  39                              DAD     SP
                                ;       ENTRY STACK POINTER IN HL FROM THE CCP
 0104'  22 0209'                        SHLD    OLDSP
                                ;       SET SP TO LOCAL STACK AREA (RESTORED AT FINIS)
 0107'  31 024B'                        LXI     SP,STKTOP
                                ;       READ AND PRINT SUCCESSIVE BUFFERS
 010A'  CD 01B9'                        CALL    SETUP   ;SET UP INPUT FILE
 010D'  FE FF                           CPI     255     ;255 IF FILE NOT PRESENT
 010F'  C2 011B'                        JNZ     OPENOK  ;SKIP IF OPEN IS OK
                                ;
                                ;       FILE NOT THERE, GIVE ERROR MESSAGE AND RETURN
 0112'  11 01E7'                        LXI     D,OPNMSG








        MACRO-80 3.33   12-Sep-79       PAGE    1-1

 0115'  CD 0194'                        CALL    ERR
 0118'  C3 0151'                        JMP     FINIS   ;TO RETURN
                                ;
                                OPENOK: ;OPEN OPERATION OK, SET BUFFER INDEX TO END
 011B'  3E 80                           MVI     A,80H
 011D'  32 0207'                        STA     IBP     ;SET BUFFER POINTER TO 80H
                                ;       HL CONTAINS NEXT ADDRESS TO PRINT
 0120'  21 0000                         LXI     H,0     ;START WITH 0000
                                ;
                                GLOOP:
 0123'  E5                              PUSH    H       ;SAVE LINE POSITION
 0124'  CD 019A'                        CALL    GNB
 0127'  E1                              POP     H       ;RECALL LINE POSITION
 0128'  DA 0151'                        JC      FINIS   ;CARRY SET BY GNB IF END FILE
 012B'  47                              MOV     B,A
                                ;       PRINT HEX VALUES
                                ;       CHECK FOR LINE FOLD
 012C'  7D                              MOV     A,L
 012D'  E6 0F                           ANI     0FH     ;CHECK LOW 4 BITS
 012F'  C2 0144'                        JNZ     NONUM
                                ;       PRINT LINE NUMBER
 0132'  CD 016A'                        CALL    CRLF
                                ;
                                ;       CHECK FOR BREAK KEY
 0135'  CD 0159'                        CALL    BREAK
                                ;       ACCUM LSB = 1 IF CHARACTER READY
 0138'  0F                              RRC             ;INTO CARRY
 0139'  DA 0151'                        JC      FINIS   ;DON'T PRINT ANY MORE
                                ;
 013C'  7C                              MOV     A,H
 013D'  CD 0187'                        CALL    PHEX
 0140'  7D                              MOV     A,L
 0141'  CD 0187'                        CALL    PHEX
                                NONUM:
 0144'  23                              INX     H
 0145'  3E 20                           MVI
 0147'  CD 0161'                        CALL
 014A'  78
 014B'  CD 0187'
 014E'  C3 0123'


 0151'  CD 016A'
 0154'  2A 0209'
```

Figure-4.2.30　[F]パラメータを付けた場合と，付けない場合の印字結果(右は付けなかった場合，左は付けた場合)

実習14 [Gn]パラメータ

現在のユーザー・エリアへ，他のユーザー・エリアから，PIPによるファイルのコピーを可能にします．但し，現在のユーザー・エリアに，PIPプログラム（PIP．COM）が存在することが前提です．

実習例として，ユーザー・エリア5に〝PIP．COM〟を作り，その上でユーザー・エリア0から，エディタ・プログラムの〝ED．COM〟をコピーし，その確認をしてみましょう．

```
A>DDT PIP.COM ↲ -----DDTによりPIP.COMをTPAにロードする.
DDT VERS 2.2
NEXT  PC
1E00 0100
-^C ---------DDTを終る.
A>USER 5 ↲ ----- ユーザー・エリア5に移行.
A>DIR ↲ --------ディレクトリを見る.
NO FILE
A>SAVE 29 PIP.COM ↲ -----TAPのPIP.COMをセーブする.
A>DIR ↲ ------------もう一度ディレクトリを見る.
A: PIP      COM ------PIPがセーブされている.
A>PIP A:=ED.COM ↲ ----ED.COMをコピーする.

INVALID FORMAT: ED.COM -----エラー・メッセージが出た.

A>PIP A:=ED.COM[G0] ↲ ------[G0] を付けてコピーする. これはOK.

A>DIR ↲ -------ディレクトリを見る. エリア0からED.COMがコピーされた.
A: PIP      COM : ED       COM
A>
```

Figure-4.2.31　[Gn] パラメータによる他のエリアからのファイルのコピー，任意のユーザー・エリアにファイルをコピーする場合は，このような手順により行う．

実習15 [H]パラメータ

インテルHEX形式のデータの転送の際，送信・受信どちらの場合でも，データのチェックを行い，チェックサムを含めて，HEX形式と合致しないデータであれば，エラー・メッセージを出力します．

```
A>TYPE DUMP.HEX ↲

:10010000210000392215023157O2CDC101FEFFC284
:100110001B0111F301CD9C01C351013E8032130023A
:1001200021000OE5CDA201E1DA5101477DE60FC2D1
:100130004401CD7201CD59010FDA51017CCD8F01FF
:100140007DCD8F01233E20CD650178CD8F01C32366
:1001500001CD72012A1502F9C9E5D5C50E0BCD05F1
:100160000OC1D1E1C9E5D5C50E025FCD0500C1D101
:10017000E1C93E0DCD65013E0ACD6501C9E60FFE20
:100180000OAD28901C630C38B01C637CD6501C9F5D6
:100190000OF0F0F0FCD7D01F1CD7D01C90E09CD05EA
:1001A0000OC93A1302FE80C2B301CDCE01B7CAB373
:1001B000137C95F16003C3213022180001 97EB757
:1001C000C9AF327C00115C000E0FCD0500C9E5D52A
:1001D000C5115C000E14CD0500C1D1E1C946494CE2
:1001E000452044554D502056455253494F4E2031DD
:1001F0002E34240D0A4E4F20494E50555420464966
:100200004C4520505245534454E54204F4E204449B2
:03021000534B2429
:0000000000

A>
```

注目

Figure-4.2.32　正しい HEX ファイル〝DUMP．HEX″を，TYPE コマンドでタイプアウトしたもの．この HEX ファイルは，ダンプ・プログラムのソース・ファイル〝DUMP．ASM″をアセンブルして生成される．

上の〝DUMP．HEX″の5ライン目の終りの方にある〝…01　C3　23………″の〝C3″を，エディタを使って〝CC″に変更し，わざとエラーのある HEX ファイルを作り，これを〝ERDUMP．HEX″としましょう．

```
A>PIP PUN:=ERDUMP.HEX[E]↲

:10010000210000392215023157O2CDC101FEFFC284
:100110001B0111F301CD9C01C351013E8032130023A
:1001200021000OE5CDA201E1DA5101477DE60FC2D1
:100130004401CD7201CD59010FDA51017CCD8F01FF
:100140007DCD8F01233E20CD650178CD8F01CC2366
:1001500001CD72012A1502F9C9E5D5C50E0BCD05F1
:100160000OC1D1E1C9E5D5C50E025FCD0500C1D101
:10017000E1C93E0DCD65013E0ACD6501C9E60FFE20
:100180000OAD28901C630C38B01C637CD6501C9F5D6
:100190000OF0F0F0FCD7D01F1CD7D01C90E09CD05EA
:1001A0000OC93A1302FE80C2B301CDCE01B7CAB373
:1001B000137C95F16003C3213022180001 97EB757
:1001C000C9AF327C00115C000E0FCD0500C9E5D52A
:1001D000C5115C000E14CD0500C1D1E1C946494CE2
:1001E000452044554D502056455253494F4E2031DD
```

エラー箇所．〝C3″であるべきものが〝CC″となっている．

```
:1001F0002E34240D0A4E4F20494E50555420464966
:100200004C45205052455354544E54204F4E204449B2
:03021000534B2429
:0000000000

A>
```

Figure-4.2.33　エラーのある HEX ファイル "ERDUMP．HEX" をパンチ・デバイスに出力する．分かり易いように[E]パラメータを付けて，スクリーンにも表示させている．何事もなく最後まで出力された．

```
A>PIP PUN:=ERDUMP.HEX[HE] ↓

:10010000210000392215023157O2CDC101FEFFC284
:100110001B0111F301CD9C01C351013E803213023A
:10012000210000E5CDA201E1DA5101477DE60FC2D1
:100130004401CD7201CD59010FDA51017CCD8F01FF ── 注目.
:100140007DCD8F01233E20CD650178CD8F01CC2366
CHECKSUM ERROR

:100140007DCD8F01233E20CD650178CD8F01CC2366 ←── エラーのあるブロックが表示される.
CORRECT ERROR, TYPE RETURN OR CTL-Z
^Z
A>
```

Figure-4.2.34　今度は [H] パラメータを付けて実行する．上と同様に [E] パラメータも付けて [HE] と複合コマンドにして実行した．このようにチェックサム・エラーを検出してそのラインを表示し，パンチ・デバイスへの送信を停止した．

このように[H]パラメータにより，HEX ファイルのエラーを検出して一旦停止します．この場合，エラー・メッセージに表示されている "RETURN" をキーインすれば，エラーを無視して転送を継続し，Ctrl-Z をキーインすれば PIP を終了します．"CORRECT ERROR" と表示されていますが，この例ではソースがディスク・ファイルなので，次の例で述べるようにエラーを訂正して，そのまま転送を継続するわけにはいきません．

```
A>PIP RDUMP.HEX=RDR:[H] ↲

CHECKSUM ERROR

:100140007DCD8F01233E20CD650178CD8F01CC2366
CORRECT ERROR, TYPE RETURN OR CTL-Z
^Z ------ この場合，Ctrl-ZでPIPを打ち切った．
A>
```

Figure-4.2.35　受信の際も［H］パラメータは有効である．リーダ・デバイスから HEX データを入力し，〝RDUMP. HEX″としてディスクにセーブする例. 入力データは先程の〝ERDUMP. HEX″を使った．

受信の場合もこのようにエラーを検出します．この場合は，RDR：デバイスが テープ・リーダに類するようなものであれば，テープの誤りを訂正した後，少しもどした適当な位置にセットして，リターンをキーインすれば継続して正しいデータが自動的に編集されて受信できます． Ctrl-Z をキーインすれば PIP を終了します．

実習16　[I]パラメータ

前述の［H］パラメータの機能に，HEX 形式のヌルレコード（：00）を削除する機能を追加したもので，送信・受信どちらの場合でも有効です．通常は紙テープからの入力の際，テープ・リーダ部などの，不要な〝00″データを無視するために使います．

```
A>PIP CON:=DUMP.HEX ↲

:10010000210000392215023157O2CDC101FEFFC284
:100110001B0111F301CD9C01C351013E803213023A
:10012000210000E5CDA201E1DA5101477DE60FC2D1
:100130004401CD7201CD59010FDA51017CCD8F01FF
      .
      .
      .
:1001F0002E34240D0A4E4F20494E50555420464966
:100200004C452050524553454E54204F4E204449B2
:03021000534B2429
:0000000000

A>
```

Figure-4.2.36　［I］パラメータなしのコンソール・デバイスへの出力例．最後のラインの：000……に注目．

```
A>PIP CON:=DUMP.HEX[I] ↲

:1001000021000039221502315702CDC101FEFFC284
:100110001B0111F301CD9C01C351013E803213023A
:1001200021000E5CDA201E1DA5101477DE60FC2D1
:100130004401CD7201CD59010FDA51017CCD8F01FF
     .
     .
     .
:1001F0002E34240D0A4E4F20494E50555420464966
:100200004C4520505245534E54204F4E204449B2
:03021000534B2429

A>
```

Figure-4.2.37　先の例にパラメータを付けた例. 最後のラインにあった：000……が送信されていないことに注目.

実習17　[L]パラメータ

転送データ（対象はアスキー・データ）の，すべての大文字を小文字に変換します．送信・受信どちらの場合にも有効です．

```
A>PIP CON:=BIOS.ASM ↲

;       MDS-800 I/O Drivers for CP/M 2.2
;       (four drive single density version)
;
;       Version 2.2 February, 1980
;
vers    equu    22      ;version 2.2
;
;       Copyright (c) 1980
;       Digital Research
;       Box 579, Pacific Grove
;       California, 93950
;
;
true    equ     0ffffh  ;value of "true"
false   equ     not true        ;"false"
test    equ     false   ;true if test bios
        .
        .
        .
A>
```

Figure-4.2.38　[L] パラメータなしで，アセンブリ・ソース・ファイル〝BIOS．ASM〞をコンソール・デバイスに出力する．

```
A>PIP CON:=BIOS.ASM[L] ↲

;       mds-800 i/o drivers for cp/m 2.2
;       (four drive single density version)
;
;       version 2.2 february, 1980
;
vers    equ     22      ;version 2.2
;
;       copyright (c) 1980
;       digital research
;       box 579, pacific grove
;       california, 93950
;
;
true    equ     0ffffh  ;value of "true"
false   equ     not true        ;"false"
test    equ     false   ;true if test bios
    .
    .
    .
A>
```

Figure-4.2.39　[L] のパラメータを付けて同様に実行する．大文字であったのが，小文字に変換されている．

実習18 [N]パラメータ

転送データ（アスキー・データが対象）にラインNo.を付けます．[N] と [N2] パラメータにより，2種類のラインNo.の表示法があります．送信・受信どちらの場合にも有効です．

```
A>PIP CON:=DUMP.ASM[N] ↲

     1: ;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
     2: ;
     3: ;       COPYRIGHT (C) 1975, 1976, 1977, 1978
     4: ;       DIGITAL RESEARCH
     5: ;       BOX 579, PACIFIC GROVE
     6: ;       CALIFORNIA, 93950
     7: ;
     8:         ORG     100H
     9: BDOS    EQU     0005H   ;DOS ENTRY POINT
    10: CONS    EQU     1       ;READ CONSOLE
    11: TYPEF   EQU     2       ;TYPE FUNCTION
    12: PRINTF  EQU     9       ;BUFFER PRINT ENTRY
    13: BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
    14: OPENF   EQU     15      ;FILE OPEN
```

```
    15: READF    EQU      20        ;READ FUNCTION
    16: ;
    17: FCB      EQU      5CH       ;FILE CONTROL BLOCK ADDRESS
    18: BUFF     EQU      80H       ;INPUT DISK BUFFER ADDRESS
     .
     .
     .
     .
     .
A>
```

Figure-4.2.40 [N] パラメータを付けてダンプ・プログラムのソース・ファイル "DUMP. ASM" をコンソール・デバイスへ出力する.

```
A>PIP CON:=DUMP.ASM[N2] ↲

000001  ;         FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
000002  ;
000003  ;         COPYRIGHT (C) 1975, 1976, 1977, 1978
000004  ;         DIGITAL RESEARCH
000005  ;         BOX 579, PACIFIC GROVE
000006  ;         CALIFORNIA, 93950
000007  ;
000008            ORG     100H
000009  BDOS      EQU     0005H    ;DOS ENTRY POINT
000010  CONS      EQU     1        ;READ CONSOLE
000011  TYPEF     EQU     2        ;TYPE FUNCTION
000012  PRINTF    EQU     9        ;BUFFER PRINT ENTRY
000013  BRKF      EQU     11       ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
000014  OPENF     EQU     15       ;FILE OPEN
000015  READF     EQU     20       ;READ FUNCTION
000016  ;
000017  FCB       EQU     5CH      ;FILE CONTROL BLOCK ADDRESS
000018  BUFF      EQU     80H      ;INPUT DISK BUFFER ADDRESS
   .
   .
   .
   .
   .
A>
```

Figure-4.2.41 [N2] パラメータを付けて同様に実行した例. 前置の0が付いたラインNo.が付けられている.

実習19　[O]パラメータ

転送の際，〝EOF″ (End Of File)を示すCtrl-Z(1AH)によるターミネート処理を無視します．[O]パラメータを付けることにより送信の場合は，データ中のCtrl-Zによる PIP の終了が起らず，アスキー・データ以外のオブジェクト・データなどの送信を行うことが可能となります．ディスク上の〝COM″ファイルを送信する場合は，自動的にこの[O]パラメータが付けられます．受信の場合は，データ中の Ctrl-Z を受信しても PIP を終了せず，アスキー・データでなくても受信を続けることが可能となります．

オブジェクト・ファイルの〝ED．COM″を送信する実習を行ってみましょう．〝OUT：″装置に送り出します．〝OUT：″装置は，後述の「PIP の特別デバイス」の項で解説しますが，筆者の使用している PIP プログラムには，この〝OUT：″デバイスに，実験のために次の機能を持たせてあります．「〝OUT：″デバイスに対して出力されたデータは，アドレス2000Hからのメモリにすべて格納されて行く，その最終データの次のアドレスには，マーカーとして必ず FFH が書き込まれる」．というものです．

```
A>DDT ED.COM ↲
DDT VERS 2.2
NEXT  PC
1B00 0100
-D100,1AFF ↲
0100 C3 C0 01 20 43 4F 50 59 52 49 47 48 54 20 28 43 ... COPYRIGHT (C
0110 29 20 31 39 37 39 2C 20 44 49 47 49 54 41 4C 20 ) 1979, DIGITAL
0120 52 45 53 45 41 52 43 48 20 44 49 53 4B 20 4F 52 RESEARCH DISK OR
0130 20 44 49 52 45 43 54 4F 52 59 20 46 55 4C 4C 24  DIRECTORY FULL$
0140 46 49 4C 45 20 45 58 49 53 54 53 2C 20 45 52 41 FILE EXISTS, ERA
0150 53 45 20 49 54 24 4E 45 57 20 46 49 4C 45 24 2A SE IT$NEW FILE$*
0160 2A 20 46 49 4C 45 20 49 53 20 52 45 41 44 2F 4F * FILE IS READ/O
0170 4E 4C 59 20 2A 2A 24 22 53 59 53 54 45 4D 22 20 NLY **$"SYSTEM"
0180 46 49 4C 45 20 4E 4F 54 20 41 43 43 45 53 53 49 FILE NOT ACCESSI
0190 42 4C 45 24 42 41 4B 24 24 24 42 41 4B 24 24 24 BLE$BAK$$$BAK$$$
01A0 2D 28 59 2F 4E 29 3F 24 4E 4F 20 4D 45 4D 4F 52 -(Y/N)?$NO MEMOR
01B0 59 24 42 52 45 41 4B 20 22 24 22 20 41 54 20 24 Y$BREAK "$" AT $
01C0 31 6D 1A 01 4D 1D 11 06 00 CD F2 19 22 6D 1A 0E 1m..M......."m..
01D0 0A CD D7 19 2B EB 21 38 1B 73 7E 3C 4F 06 00 60 ....+.!8.s~<O..`
01E0 69 0E 08 CD CD 19 22 39 1B 11 00 04 2A 39 1B 19 i....."9....*9..
   .
   .
   .
   .
   .
1A00 13 1A 9C 67 C9 5F 16 00 7B 96 5F 7A 23 9E 57 EB ...g._..{._z#.W.
1A10 C9 06 0C 48 0D C2 14 1A 3D C2 13 1A C9 00 00 00 ...H....=.......
1A20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1A30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

注目．（0100C0 行の 1A）

```
1A40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1A50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1A60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1A70 00 00 00 00 20 20 20 20 20 20 20 20 4C 49 42 00 ....        LIB.
1A80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1A90 00 00 00 00 00 00 58 24 24 24 24 24 24 24 4C 49 ......X$$$$$$$LI
1AA0 42 00 00 00 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A B...............
1AB0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
1AC0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
1AD0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
1AE0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
1AF0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
-F100,8000,00 ↲
-^C
A>
```

Figure-4.2.42 アドレス1C2Hに現れる最初の〝1AH″に注目しておく．データの最後は，アドレス1AFFHで終っていることにも注目．確認し終ったら，100H～8000Hぐらいをゼロクリアしておく．

次に〝ED．COM″を〝ED．TST″とでもリネームしておきます．エクステンションが〝COM″であるファイルは，送信の場合，自動的に[O]パラメータが付いてしまうので，それを実習では避けたいために行います．

```
A>REN ED.TST=ED.COM ↲
A>
```

Figure-4.2.43
実習を分かりやすくするために，〝COM″以外のエクステンションにリネームしておく．

これで準備が整いました．PIP を実行してみます．

```
A>PIP OUT:=ED.TST ↲

A>
```

Figure-4.2.44
[O] パラメータなしで，オブジェクト・ファイルの〝ED．TST″を〝OUT：″デバイスに送り出す．筆者の〝OUT：″デバイスは，データをアドレス2000Hからのメモリに格納して行く．PIP が終了していることに注目．

```
A>DDT ↲
DDT VERS 2.2
-D2000,4000 ↲
2000 C3 C0 01 20 43 4F 50 59 52 49 47 48 54 20 28 43 ... COPYRIGHT (C
2010 29 20 31 39 37 39 2C 20 44 49 47 49 54 41 4C 20 ) 1979, DIGITAL
2020 52 45 53 45 41 52 43 48 20 44 49 53 4B 20 4F 52 RESEARCH DISK OR
2030 20 44 49 52 45 43 54 4F 52 59 20 46 55 4C 4C 24  DIRECTORY FULL$
2040 46 49 4C 45 20 45 58 49 53 54 53 2C 20 45 52 41 FILE EXISTS, ERA
2050 53 45 20 49 54 24 4E 45 57 20 46 49 4C 45 24 2A SE IT$NEW FILE$*
2060 2A 20 46 49 4C 45 20 49 53 20 52 45 41 44 2F 4F * FILE IS READ/O
2070 4E 4C 59 20 2A 2A 24 22 53 59 53 54 45 4D 22 20 NLY **$"SYSTEM"
2080 46 49 4C 45 20 4E 4F 54 20 41 43 43 45 53 53 49 FILE NOT ACCESSI
2090 42 4C 45 24 42 41 4B 24 24 24 42 41 4B 24 24 24 BLE$BAK$$$BAK$$$
20A0 2D 28 59 2F 4E 29 3F 24 4E 4F 20 4D 45 4D 4F 52 -(Y/N)?$NO MEMOR
20B0 59 24 42 52 45 41 4B 20 22 24 22 20 41 54 20 24 Y$BREAK "$" AT $
20C0 31 6D FF 00 00 00 00 00 00 00 00 00 00 00 00 00 1m..............
20D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
20E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
20F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
2100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
      .
      .
      .
      .
      .
```

マーカー．ここで転送が打ち切られている．

Figure-4.2.45　終了したPIPの結果を確認する．DDTを起動して，アドレス2000Hからダンプしてみる．

DDTによるダンプでわかるように，PIPによる転送はアドレス20C1Hで止っています．次のデータ〝FFH〟は，最後のデータを示す〝OUT：〟デバイスのマーカーです．

このダンプの内容とFigure-4.2.41のオリジナル・ファイルのダンプの内容を比較して下さい．転送された最後のデータ〝…31，6D〟の後には〝1AH〟が来ていたことが分かります．

このようにPIPによる転送は，〝COM〟ファイル以外（〝COM〟ファイルは自動的に［O］パラメータが付く）のデータであれば，［O］パラメータを付けないとき，転送データ中に〝1AH〟があると，その時点でPIPを終了します．送信・受信，どちらの場合も終了します．これでは〝1AH〟を含む可能性のあるオブジェクト・ファイルやデータ・ファイルなどは転送できないことになります．

次に［O］パラメータを付けて実行してみましょう．

```
A>PIP OUT:=ED.TST[O] ↲

A>
```

Figure-4.2.46
［O］パラメータを付けて，ファイル〝ED．TST〟を〝OUT：〟デバイスに送り出す．前と同様にPIPが終了した．

```
A>DDT ↲
DDT VERS 2.2
-D2000,3AFF ↲
2000 C3 C0 01 20 43 4F 50 59 52 49 47 48 54 20 28 43 ... COPYRIGHT (C
2010 29 20 31 39 37 39 2C 20 44 49 47 49 54 41 4C 20 ) 1979, DIGITAL 
2020 52 45 53 45 41 52 43 48 20 44 49 53 4B 20 4F 52 RESEARCH DISK OR
2030 20 44 49 52 45 43 54 4F 52 59 20 46 55 4C 4C 24  DIRECTORY FULL$
2040 46 49 4C 45 20 45 58 49 53 54 53 2C 20 45 52 41 FILE EXISTS, ERA
2050 53 45 20 49 54 24 4E 45 57 20 46 49 4C 45 24 2A SE IT$NEW FILE$*
2060 2A 20 46 49 4C 45 20 49 53 20 52 45 41 44 2F 4F * FILE IS READ/O
2070 4E 4C 59 20 2A 2A 24 22 53 59 53 54 45 4D 22 20 NLY **$"SYSTEM" 
2080 46 49 4C 45 20 4E 4F 54 20 41 43 43 45 53 53 49 FILE NOT ACCESSI
2090 42 4C 45 24 42 41 4B 24 24 24 42 41 4B 24 24 24 BLE$BAK$$$BAK$$$
20A0 2D 28 59 2F 4E 29 3F 24 4E 4F 20 4D 45 4D 4F 52 -(Y/N)?$NO MEMOR
20B0 59 24 42 52 45 41 4B 20 22 24 22 20 41 54 20 24 Y$BREAK "$" AT $
20C0 31 6D 1A 01 4D 1D 11 06 00 CD F2 19 22 6D 1A 0E 1m..M......."m..
20D0 0A CD D7 19 2B EB 21 38 1B 73 7E 3C 4F 06 00 60 ....+.!8.s~<O..`
20E0 69 0E 08 CD CD 19 22 39 1B 11 00 04 2A 39 1B 19 i....."9....*9..
     .
     .
     .
     .
     .
3900 13 1A 9C 67 C9 5F 16 00 7B 96 5F 7A 23 9E 57 EB ...g._..{._z#.W.
3910 C9 06 0C 48 0D C2 14 1A 3D C2 13 1A C9 00 00 00 ...H....=.......
3920 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3930 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3940 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3950 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3960 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3970 00 00 00 00 20 20 20 20 20 20 20 20 4C 49 42 00 ....        LIB.
3980 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3990 00 00 00 00 00 00 58 24 24 24 24 24 24 24 4C 49 ......X$$$$$$$LI
39A0 42 00 00 00 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A B...............
39B0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
39C0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
39D0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
39E0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
39F0 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
3A00 FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
-
```

Figure-4.2.47 [O] パラメータを付けて実行した結果を確認する．DDT でアドレス2000Hからダンプすると，このようにオリジナル・ファイルが完全に転送されていることが分かる．

このように，[O] パラメータを付けることにより PIP によるアスキー・データ以外の転送（送信・受信とも）が可能となります．

実習20 [Pn]パラメータ

プリント用紙にリストをとる時など，nラインごとのページ割り付けを行います．nを1または省略すると，自動的にn＝60となります．

Figure-4.2.48
DUMPプログラムのソース・ファイル"DUMP.ASM"を"LST:"デバイスへ出力する．[P40]と同時に[T8]も付けて8カラムのタブ(後述)機能を持たせた．

```
;       FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
;
;       COPYRIGHT (C) 1975, 1976, 1977, 1978
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
        ORG     100H
BDOS    EQU     0005H   ;DOS ENTRY POINT
CONS    EQU     1       ;READ CONSOLE
TYPEF   EQU     2       ;TYPE FUNCTION
PRINTF  EQU     9       ;BUFFER PRINT ENTRY
BRKF    EQU     11      ;BREAK KEY FUNCTION (TRUE IF CHAR READY)
OPENF   EQU     15      ;FILE OPEN
READF   EQU     20      ;READ FUNCTION
;
FCB     EQU     5CH     ;FILE CONTROL BLOCK ADDRESS
BUFF    EQU     80H     ;INPUT DISK BUFFER ADDRESS
;
;       NON GRAPHIC CHARACTERS
CR      EQU     0DH     ;CARRIAGE RETURN
LF      EQU     0AH     ;LINE FEED
;
;       FILE CONTROL BLOCK DEFINITIONS
FCBDN   EQU     FCB+0   ;DISK NAME
FCBFN   EQU     FCB+1   ;FILE NAME
FCBFT   EQU     FCB+9   ;DISK FILE TYPE (3 CHARACTERS)
FCBRL   EQU     FCB+12  ;FILE'S CURRENT REEL NUMBER
FCBRC   EQU     FCB+15  ;FILE'S RECORD COUNT (0 TO 128)
FCBCR   EQU     FCB+32  ;CURRENT (NEXT) RECORD NUMBER (0 TO 127)
FCBLN   EQU     FCB+33  ;FCB LENGTH
;
;       SET UP STACK
        LXI     H,0
        DAD     SP
;       ENTRY STACK POINTER IN HL FROM THE CCP
        SHLD    OLDSP
;       SET SP TO LOCAL STACK AREA (RESTORED AT FINIS)
        LXI     SP,STKTOP
;       READ AND PRINT SUCCESSIVE BUFFERS
```

```
        CALL    SETUP   ;SET UP INPUT FILE
        CPI     255     ;255 IF FILE NOT PRESENT
        JNZ     OPENOK  ;SKIP IF OPEN IS OK
;
;       FILE NOT THERE, GIVE ERROR MESSAGE AND RETURN
        LXI     D,OPNMSG
        CALL    ERR
        JMP     FINIS   ;TO RETURN
;
OPENOK: ;OPEN OPERATION OK, SET BUFFER INDEX TO END
        MVI     A,80H
        STA     IBP     ;SET BUFFER POINTER TO 80H
;       HL CONTAINS NEXT ADDRESS TO PRINT
        LXI     H,0     ;START WITH 0000
;
GLOOP:
        PUSH    H       ;SAVE LINE POSITION
        CALL    GNB
        POP     H       ;RECALL LINE POSITION
        JC      FINIS   ;CARRY SET BY GNB IF END FILE
        MOV     B,A
;       PRINT HEX VALUES
;       CHECK FOR LINE FOLD
        MOV     A,L
        ANI     0FH     ;CHECK LOW 4 BITS
        JNZ     NONUM
;       PRINT LINE NUMBER
        CALL    CRLF
;
;       CHECK FOR BREAK KEY
        CALL    BREAK
;       ACCUM LSB = 1 IF CHARACTER READY
        RRC             ;INTO CARRY
        JC      FINIS   ;DON'T PRINT ANY MORE
;
        MOV     A,H
        CALL    PHEX
        MOV     A,L
        CALL    PHEX
NONUM:
```

Figure-4.2.49 [P40]により1ページに40行の割り付けが行われたリスト

実習21 [Q文字列^Z]パラメータ
実習22 [S文字列^Z]パラメータ

〝S″パラメータは，転送データをサーチして行き，〝文字列″が見つかると実際の転送を開始（その〝文字列″を含む）します．〝Q″パラメータは，転送中のデータをサーチし〝文字列″があるとそこで転送をストップ（その〝文字列″を転送してから）し，PIPを終了します．

注）〝文字列″が小文字を含む場合は，PIPのみ先に起動しておき，PIPのプロンプト〝*″の後にコマンド・ラインを記述すること．下の例のように，1行で記述した場合は，コマンド・ラインがすべて大文字に変換されるので，小文字のサーチが不可能である．

```
A>PIP CON:=DUMP.ASM[SOPENOK:^ZQGLOOP:^Z]↲

OPENOK: ;OPEN OPERATION OK, SET BUFFER INDEX TO END
        MVI     A,80H
        STA     IBP     ;SET BUFFER POINTER TO 80H
;       HL CONTAINS NEXT ADDRESS TO PRINT
        LXI     H,0     ;START WITH 0000
;
GLOOP:

A>
```

Figure-4.2.50　〝CON:″デバイスにファイル〝DUMP.ASM″を出力する．SとQパラメータを同時に使って文字列〝OPENOK:″から〝GLOOP:″の範囲を出力させる．

実習23 [R]パラメータ

〝SYS″アトリビュートの付いているファイルを転送可能にします．同時に後述の［W］パラメータも自動的にセットされます．

```
A>STAT LOAD.COM↲

 Recs  Bytes  Ext Acc
   14     2k    1 R/W A:(LOAD.COM)
Bytes Remaining On A: 40k

A>
```

Figure-4.2.51
〝SYS″アトリビュートの付いた〝LOAD.COM″の確認．

```
A>PIP B:=LOAD.COM ↲

NO FILE: =LOAD.COM

A>
```

Figure-4.2.52
[R] パラメータなしで，このファイルをドライブB：上にコピーを試みるが，PIP は〝SYS〞ファイルを発見できない．

```
A>PIP B:=LOAD.COM[R] ↲

A>
```

Figure-4.2.53
[R] パラメータを付けて同様に試みる．ドライブB：上にコピーされて，PIP が終了している．

実習24 [Tn]パラメータ

転送データ中のタブをn文字ごとのスペースに設定します．

```
A>PIP CON:=DUMP.ASM[T20] ↲

;                   FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN
;
;                   COPYRIGHT (C) 1975, 1976, 1977, 1978
;                   DIGITAL RESEARCH
;                   BOX 579, PACIFIC GROVE
;                   CALIFORNIA, 93950
;
                    ORG                 100H
BDOS                EQU                 0005H               ;DOS ENTR......
CONS                EQU                 1                   ;READ CON......
TYPEF               EQU                 2                   ;TYPE FUN......
PRINTF              EQU                 9                   ;BUFFER P......
BRKF                EQU                 11                  ;BREAK KE......
OPENF               EQU                 15                  ;FILE OPE......
READF               EQU                 20                  ;READ FUN......

        .
        .
A>
```


Figure-4.2.54 ファイル〝DUMP．ASM〞を〝CON：〞デバイスへ，タブを20文字ごとに設定して出力する．
Figure-4.2.40～41 などの8文字ごとの場合と比較して下さい．但し〝CON：〞がスクリーンの場合，[T]パラメータを付けなくても，スクリーンの機能として8文字ごとのタブが行われる．

実習25 [U]パラメータ

アスキー・データの小文字を，すべて大文字に変換して転送します．

```
A>PIP CON:=BIOS.ASM[U] ↲

;       MDS-800 I/O DRIVERS FOR CP/M 2.2
;       (FOUR DRIVE SINGLE DENSITY VERSION)
;
;       VERSION 2.2 FEBRUARY, 1980
;
VERS    EQU     22      ;VERSION 2.2
;
;       COPYRIGHT (C) 1980
;       DIGITAL RESEARCH
;       BOX 579, PACIFIC GROVE
;       CALIFORNIA, 93950
;
;
TRUE    EQU     0FFFFH  ;VALUE OF "TRUE"
FALSE   EQU     NOT TRUE        ;"FALSE"
TEST    EQU     FALSE   ;TRUE IF TEST BIOS
;
      .
      .
      .
A>
```

Figure-4.2.55 ファイル "BIOS．ASM" 中の小文字を，大文字に変換して "CON：" デバイスに出力する．
オリジナルのリスト Figure-4.2.38と比較して下さい．

実習26 [V]パラメータ

ディスクからディスクへのファイルのコピーの際，書き込み後，即読み出してソース・データと比較し，その一致を確認しながらコピーを行わせます．コピーされた内容の保障ができます．

```
A>PIP B:=ASM.COM[V] ↲

A>
```

Figure-4.2.56
ドライブ B：上に "ASM．CON" を [V] パラメータを付けて，確実にコピーした．

実習27　[W]パラメータ

PIP 転送によりディスク上にファイルを作る時，すでにそのディスク上に存在している〝R／O″アトリビュートの付いた同名のファイル（もしそのようなファイルがあれば）の〝R／O″を無視して，新しいファイルのコピーを可能にします（旧ファイルは削除される）.

```
A>STAT B:SYSGEN.COM $R/O ↲

SYSGEN.COM set to R/O
A>
```

Figure-4.2.57
ドライブB：上の〝SYSGEN．COM″を〝R／O″ファイルに変える.

```
A>PIP B:=SYSGEN.COM ↲

DESTINATION IS R/O, DELETE (Y/N)?N
**NOT DELETED**

A>
```

Figure-4.2.58
[W]パラメータを付けずにドライブB：上に同名の〝SYSGEN．COM″をコピーする.
このようにメッセージが出力され，〝Y″をキーインしなければコピーされない.

```
A>PIP B:=SYSGEN.COM[W] ↲

A>
```

Figure-4.2.59
[W] パラメータを付けて同様に実行した.
このようにメッセージは出力されず，人が手を掛けることなくコピーされている.

実習28　[Z]パラメータ

アスキー・データの受信の際，入力データのパリティ・ビットを0にします（ビット7を0にする）.
ここでは2台の CP／M マシン（カナ文字が使えるもの）を使った実例を示しておきます.

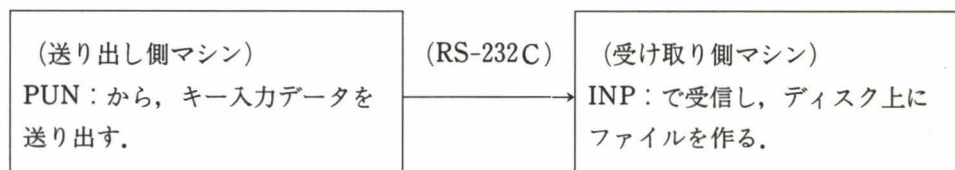
（送り出し側マシン） PUN：から，キー入力データを送り出す.	（RS-232C）→	（受け取り側マシン） INP：で受信し，ディスク上にファイルを作る.

```
A>PIP PUN:=CON:,EOF: ↲
ABCDEFG hijklmn ↲ LF
1234567890 ↲ LF
ｱｲｳｴｵｶｷｸｹｺ ↲ LF ^Z
A>
      LF＝ラインフィード．LFはCtrl-Jでも代用できる．
```

Figure-4.2.60
送り出し側マシン実行例．
キー入力が"PUN:"デバイスから送り出される．カタカナのラインに注目．Ctrl-Z をキーインすることにより，"EOF" キャラクタを送り出す．

```
A>PIP NON/Z.TST=INP: ↲

A>
```

Figure-4.2.61
［Z］パラメータなしの受信側実行例．
"INP:" デバイスからデータを受信し，ファイル "NON／Z．TST" としてディスクにセーブする．"EOF" キャラクタの受信により，PIPが終了する．

```
A>DDT NON/Z.TST ↲
DDT VERS 2.2
NEXT  PC
0180 0100
-D100,17F ↲
0100 41 42 43 44 45 46 47 20 68 69 6A 6B 6C 6D 6E 0D ABCDEFG hijklmn.
0110 0A 31 32 33 34 35 36 37 38 39 30 0D 0A B1 B2 B3 .1234567890..ｱｲｳ
0120 B4 B5 B6 B7 B8 B9 BA 0D 0A 1A 1A 1A 1A 1A 1A 1A ｴｵｶｷｸｹｺ.........
0130 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0140 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0150 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0160 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0170 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
-^C
A>
```

Figure-4.2.62　［Z］パラメータなしで作られたファイル "NON／Z．TST" をDDT でダンプして内容を確認する．
カタカナもファイルされており，ビット 7 は 0 にされていないことが分る．(ア＝B1H＝10110001)
注）カタカナ表示用に一部改造した DDT を使用．

```
A>PIP PUT/Z.TST=INP:[Z] ↲

A>
```

Figure-4.2.63
［Z］パラメータを付けた受信側実行例．
今回はファイル名を "PUT／Z．TST" として実行した．

```
A>DDT PUT/Z.TST↲
DDT VERS 2.2
NEXT  PC
0180 0100
-D100,17F↲
0100 41 42 43 44 45 46 47 20 68 69 6A 6B 6C 6D 6E 0D ABCDEFG hijklmn.
0110 0A 31 32 33 34 35 36 37 38 39 30 0D 0A 31 32 33 .1234567890..123
0120 34 35 36 37 38 39 3A 0D 0A 1A 1A 1A 1A 1A 1A 1A 456789:.........
0130 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0140 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0150 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0160 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
0170 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A 1A ................
-^C
A>
```

Figure-4.2.64　DDTによるファイル〝PUT／Z．TST〟の内容確認．
カタカナのデータであったものがビット7を0にされて，B1H→31H＝〝1〟のように変化している．このように受信データのパリティ・ビットが0にされていることが分かる．

1台のCP／Mマシン（カナ文字が使えるもの）でも，次のように［Z］パラメータの働きを確認できます．コンソールからの入力を，コンソールに出力する（要するに，キーボードからの入力を，自分自身のスクリーンへ出力する）コマンドを実行してみましょう．

```
A>PIP↲
*CON:=CON:↲ ----- パラメータなしで実行.
AABBCCDD   aabbccdd   ｱｱｲｲｳｳｴｴ↲LF^Z------ 2重になっている最初の文字がコンソール入力のもの,
                                              次がコンソール出力のもの.
*CON:=CON:[Z]↲ ------- [Z] パラメータを付けて実行.
AABBCCDD   aabbccdd   ｱ1ｲ2ｳ3ｴ4↲LF^Z

*                        "ｱ"と入力されたものが "1" と出力されていることに注目.
                               前述の例と同じ理由による.
```

Figure-4.2.65　1台のCP／Mマシンで，［Z］パラメータの働きを確認する．

PIPの特別デバイス

PIPが取り扱うことができる〝装置〟には，CON：，RDR：，PUN：，LST：の4つのロジカル・デバイスに対するそれぞれ4種類のフィジカル・デバイス，計16種（第1章の「ディスク以外の周辺装置」および第4章の「STATコマンド」参照）の他に，次に示す4種類の特別な〝装置〟があります．

EOF：ファイルの終り（End Of File）を表し，PIPにターミネート処理させる〝1AH〟のコードを送り出すロジカル上の〝装置〟．
実際の使用例であるFigure-4.2.22や，Figure-4.2.60などを参照．

NUL：40個のヌル・コード（00）を送り出すロジカル上の〝装置〟．紙テープ・パンチャなどの場合は，テープのリーダ部を付けるために使用される．OUT：デバイス（筆者のPIPコマンドには，出力するデータをアドレス2000Hからのメモリ上に格納して行く機能がパッチしてある）を使った次の実行例を参照．

```
A>PIP OUT:=CON:,NUL:,CON: ↲
ABCDEFGVWXYZ
       └^Z └^Z
A>DDT ↲
DDT VERS 2.2
-D2000 ↲
2000 41 42 43 44 45 46 47 00 00 00 00 00 00 00 00 00 ABCDEFG.........
2010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
2020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 56 ...............V
2030 57 58 59 5A FF FF FF FF FF FF FF FF FF FF FF FF WXYZ............
2040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
     .
     .
     .
     .
```

Figure-4.2.66　〝OUT:〟デバイスに，キー入力データ→40個のヌル→再びキー入力データの順に送信される．

PRN：〝LST：〟デバイスと似ているが，次の相違点がある．

- ラインNo.が付く．
- タブが8文字ごとに設定される．
- 60ラインごとのページ割り付けが行われる．

これらの機能があるため，〝PRN：〟デバイスは，各種のリストを取る時によく使われる．

INP： ユーザーが，PIP プログラム (PIP．COM) の中にパッチして組み込みが可能な，入力のための〝装置〞.

PIP プログラムの中のパッチ用のユーザー・エリアは，DDT により PIP．COM をロードした時のアドレス109H～1FFHにある.

PIP がINP：デバイスを呼ぶ場合，アドレス103HがCALL され，109Hのデータを入力データとして持って行く.

次に，筆者がいろいろな実験に使う目的で，この〝INP：〞をパッチした作業の様子を示しておきます．パッチしたルーチンの内容は前述してある通り．「ポート〝F9〞への入力データをアドレス2000Hから順次格納して行き，格納データの最後には目印として常に〝FFH〞を付ける.」というものである．ポート・アドレスは，各自のマシンにより異なりますので注意して下さい.

```
A>DDT PIP.COM ↲
DDT VERS 2.2
NEXT  PC
1E00 0100
-D100,1FF ↲   元の〝PIP.COM〞の冒頭部分をダンプしてみる.
0100 C3 CE 04 C9 00 00 C9 00 00 1A 00 00 00 00 00 00 ................
0110 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0120 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0130 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0140 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0150 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0160 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0170 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0180 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
0190 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01A0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01B0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01C0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01D0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01E0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
01F0 28 49 4E 50 3A 2F 4F 55 54 3A 53 50 41 43 45 29 (INP:/OUT:SPACE)
-A103 ↲
0103  JMP 110 ↲
0106  ↲
-S10E ↲
010E 00 ↲     }
010F 00 20 ↲  } このアドレスに値2000Hを書き込んでおく.
0110 28 . ↲
-A110 ↲
0110  IN F8 ↲
0112  ANI 40 ↲    それぞれのマシンによって，ポートアドレスなどが異なるので注意.
0114  JZ 110 ↲
0117  IN F9 ↲
0119  STA 109 ↲
011C  LHLD 10E ↲
011F  MOV M,A ↲
```

```
0120   INX H↲
0121   MVI M,FF↲
0123   SHLD 10E↲
0126   RET↲
0127   ↲
-D100,12F↲
0100 C3 CE 04 C3 10 01 C9 00 00 1A 00 00 00 00 00 20 ................
0110 DB F8 E6 40 CA 10 01 DB F9 32 09 01 2A 0E 01 77 ...@.....2..*..w
0120 23 36 FF 22 0E 01 C9 55 54 3A 53 50 41 43 45 29 #6."...UT:SPACE)
-^C
A>SAVE 29 PIP.COM ↲
A>
```

Figure-4.2.67　INP：デバイスのパッチ作業

OUT： "INP：" と同様に，PIP プログラムの中にパッチして組み込み可能な "装置" である．PIPが "OUT：" を呼ぶと，レジスタCに出力データを持ってアドレス106Hが CALL される．

次に筆者用に，"INP："と同じようにパッチした部分のデータを示しておきます．このルーチンの機能は，"OUT：" デバイスから出力されるデータが "INP：" の場合と同様に，アドレス2000Hから格納されて行く（転送データをメモリ上に出力する実験用の機能）．

```
   .
   .
   .                                        "20" に変更しておくこと.
-D100,10F↲
0100 C3 CE 04 C3 10 01 C3 80 01 1A 00 00 00 00 00 [20] ................
-D180,18F↲
0180 2A 0E 01 71 23 36 FF 22 0E 01 C9 50 41 43 45 29 *..q#6."...PACE)
-L180,18A↲
  0180  LHLD 010E
  0183  MOV  M,C
  0184  INX  H
  0185  MVI  M,FF
  0187  SHLD 010E
  018A  RET
  018B
-
   .                             このパッチは，マシンに依存する部分がないので，すべての
   .                             CP/Mでそのまま使用できる．
   .
```

Figure-4.2.68　OUT：デバイスのパッチ作業

4.3　ED　（テキスト・エディタ）

——EDitor——

コマンド形式・機能

ED␣x : filename.ext

〝filename.ext〞が既存ファイルであれば，そのファイルに対してエディタを起動する．新ファイルであれば，新ファイルを新たに作り，エディタを起動する．いずれもドライブｘ：上で実行される．

注）　ドライブ名ｘ：はそれがログイン・ディスクの場合，省略できる．

ED内コマンドの機能

アペンド

- nA ……………エディット・バッファに，ディスクからテキストをｎ行ロードする．
- 0A ……………エディット・バッファの容量の半分まで，ディスクからテキストをロードする．

CPの移動

- ±B ……………CP をバッファ内テキストの先頭(＋)／最後(－)にセットする．
- ±nL …………±ｎ行移動し，その行の頭に CP をセットする．
- ±n ……………±n行移動し，その行の頭にCPをセットし，その行をタイプアウトする(＝±nLT)．
- 0 ………………上記〝±n〞のｎ＝０の場合．CP をその行の頭にセットし，その行をタイプアウトする（＝０LT)．
- ↲ ………………CP を次の行の頭にセットしてその行をタイプアウトする（=LT)．
- n：……………CP をライン No. ｎ の頭にセットする．
- ±nC …………CP を±ｎ文字移動する．
- nF文字列 ……サーチの欄参照．
- nN文字列 ……　〃

タイプアウト

- ±nT …………CP から±n行分タイプアウトする．
- 0T ……………行の頭から CP までをタイプアウトする．
- n：：mT ……ライン No. n～m の間の行をタイプアウトする．
- ±nP …………CP から±ｎページ分（１ページは23行）をタイプアウトする．
- ±n／0／↲ …………「CP の移動」欄参照．

サーチ

nF文字列 ……CP以後，バッファ内でn番目に出合った〝文字列〟の最後にCPをセットする．

nN文字列 ……バッファ内に限らず，CP以後，ディスク上の未アペンドの行も自動的にアペンドし，n番目に出合った〝文字列〟の最後にCPをセットする．

削除

±nD …………CPから±n文字分を削除する．

±nK …………CPの行以後（CPの行を含む）／以前（含まない）のn行分を削除する．

nJ文字列1^Z文字列2^Z文字列3 ……「置き替え・変更」の欄参照．

挿入

I ………………CP以後からインサート可能なインサート・モードに入る．〝I〟の終了は〝Ctrl-Z〟で．

I文字列^Z …CP以後に〝文字列〟を挿入する．

I文字列⏎ ……CP以後に〝文字列〟を挿入し，そこまでを新しい行としてCR／LFを挿入する．

nJ文字列1^Z文字列2^Z文字列3 ……「置き替え・変更」の欄参照．

R ……………… 〝nX〟によりテンポラリ・ファイル〝X$$$$$$$.LIB〟にセーブされている行を，CP以後に挿入する．

Rfilename ……エクステンションが〝LIB〟であるライブラリ・ファイルをCP以後に挿入する．（ライブラリ・ファイルは，オリジナル・ソース・ファイルと同一ディスク上または，ニューファイルが作られるディスク上になくてはならない．）

置き替え・変更

nS文字列1^Z文字列2

CP以後，バッファ内で〝文字列1〟を〝文字列2〟に置き替え，CPを〝文字列2〟の最後にセットすることをn回くり返す．

nJ文字列1^Z文字列2^Z文字列3

CP以後，バッファ内で最初に出合った〝文字列1〟のあとに〝文字列2〟を挿入し，その後〝文字列3〟に出合うまでのすべての文字を削除する（文字列3は残る）．この動作を同一ループ内でn回くり返す．

nX ……………CPからn行分をXコマンド用テンポラリ・ファイルに〝X$$$$$$$.LIB〟としてセーブする．〝nX〟を実行するごとにセーブされた内容は継ぎ足されて行く．

R ………………上記ファイル〝X$$$$$$$.LIB〟をCP以後に挿入する．

0X ……………上記ファイル〝X$$$$$$$.LIB〟を空にする．

±U ……………〝U〟の実行以後入力された文字は，すべて大文字に変換される．〝−U〟で〝U〟がキャンセルされる．

マクロ

nMcommand… 〝command〟をバッファの最後まで，n回くり返す．

Mcommand… 〝command〟をバッファの最後がくるまで可能な限りくり返す（0M，1Mと同じ）．

ライン No.

±V……………"－V" でラインNo.が表示されないモードになる．"V" で再び表示される．(Version 1.4 では，"－V" モードで ED が起動するので，"V" によりラインNo.を表示させると便利である．)

サイズ バッファ

0V ……………エディット・バッファの「空エリア／バッファ全体のサイズ」を表示する．

ディレー

nZ ……………n＝1の時，約1/4秒（4 MHz CPU クロック時）のディレー・タイムをとる．コンソールの表示を遅くできる．

セーブ・ターミネート

nW ……………エディット・バッファの最初からn行を，テンポラリ・ファイル"filename．$$$" にセーブする．セーブされたn行は，エディット・バッファから削除される．

E………………エディット・バッファのテキストと，残りのソース・ファイルをディスクにセーブし，ED を終了し CP／M にもどる．

H………………上記 "E" コマンドを実行した後，再び同一ファイル名で EDを起動する．今まで行ってきたエディット作業の内容を安全を期すために，オリジナル・ファイルに組み入れながら ED を続行できる．

O………………今，行っているすべてのことを，キャンセル，クリアし ED が起動した時の状態にもどす．

Q………………ED を中止して，CP／M にもどる．オリジナル・ファイルは何も変更されない．

その他

n : command … CP をラインnの頭にセットし，"command" を実行する．

: n command …"command" を CP からラインnまでについて実行する．

注）○CP はキャラクタ・ポインタの略

○± の＋記号は省略できる．

○ nが1の時は1を省略できる．"#" 記号を n＝65535（最大数）として使用できる．

○各コマンドは並べて記述できる．

○ I，S，J，R，Fなどのコマンドを大文字で与えた場合，小文字を含むテキストは小文字の部分が，大文字に変換されてしまう．よって，小文字を含むテキストのエディットには，これらのコマンドは小文字で与えること．

○ED 内で，コントロール・キーによるライン・エディッティング機能が使用できます（「実習のはじめに」参照)．

さらに　Ctrl-L により，"♪" を使わずに CR／LF を文字列の中に挿入することができます．

実習1　新ファイルの作成

```
A>ED ABCD.XYZ↲

NEW FILE
     : *i↲ ------ iコマンドでインサート・モードに入る。小文字のiに注目。
    1:  Type text lines . . . . ↲
              .
          (  Insert text  ) ------各行の終りは↲をキーインする。
              .
     :^Z  ------- Ctrl-Zでインサート・モードを終る。
     : *     ミスタイプや変更したいことがあれば編集する。
              .
              .
     : *E↲ ------EDを終了する。

A> -------CP/Mに戻った。新ファイル "ABCD.XYZ" が作成されている。
```

Figure-4.3.1　新しく作成するファイルを "ABCD．XYZ" としてEDを起動する．"I" コマンドでインサート・モードに入る．"I" を大文字で与えると，小文字でキーインしたテキストが大文字に変換されてしまうので注意．

実習2　既存ファイルのエディット

第5章の「CP／Mによるマシン語開発実習」で作成するアセンブリ・ソース・ファイルの "SORT．ASM" を題材として，ED内のすべてのコマンドの使い方を実例で解説します．実際のコマンドでは，複数のコマンドを並べて記述しているものが多いので，どのようなコマンドが組み合わされているのか注意して見て下さい．

例えば "BFEQU^Z0LT" ⇒ B＋F＋0L＋Tの組み合わせ．
リスト中の "^Z" は "Ctrl-Z" を示します．

```
 1:  ;=================================================================
 2:  ;               SORT PROGRAM for CP/M Learning System #2
 3:  ;    This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
 4:  ;====================================================== by Y.MURASE ===
 5:
 6:            ORG     100H
 7:  DATTOP    EQU     4000H           ;DATA AREA TOP ADDRESS
 8:  DATEND    EQU     4FFFH           ;DATA AREA END ADDRESS
 9:  ENDADR    EQU     (DATEND+1) SHR 8     ;HIGH 8 BIT OF NEXT '4FFF' = '50'
10:  BDOS      EQU     0005H           ;SYSTEM CALL ENTRY POINT
11:
12:            LXI     D,MSGSTRT       ;)
13:            MVI     C,9             ;)
14:            CALL    BDOS            ;) START MESSAGE OUT SYSTEM CALL
15:
16:            LXI     D,DATTOP        ;SET TOP ADR IN D,E
17:            LXI     H,DATTOP        ;SET TOP ADR IN H,L
18:            MVI     C,0             ;SET 0 IN C
19:  NEXT1:
20:            MOV     A,M             ;GET DATA
21:            CMP     C               ;COMPARE WITH C
22:            JZ      SORT            ;IF A=C JUMP SORT:
23:  ;                                *THIS STEP NOT A=C, GO NEXT DATA
24:            INX     H               ;SET HL FOR NEXT DATA ADR
25:  ;                                *CHECK OF DATA END
26:  NEXT2:    MOV     A,H             ;GET HIGH 8 BIT OF NEXT DATA ADR
27:            CPI     ENDADR          ;COMRARE WITH HIGH 8 BIT OF BUF END ADR
28:            JNZ     NEXT1           ;IF NOT END, GO NEXT DATA
29:  ;                                *THIS STEP END OF DATA BUFFER
30:            INR     C               ;SET NEXT PATTERN
31:            JZ      DONE            ;IF C=0 JUMP PROGRAM END
32:  ;                                *THIS STEP C=NEXT PATTERN, CONTINUE
33:            MOV     H,D
34:            MOV     L,E             ;)ADJUST D,E = H,L
35:            JMP     NEXT1           ;C=NEXT PATTERN AND GO LOOP
36:
37:  ;                                *THIS STEP A=C, SORT THE DATA
38:  SORT:
39:            LDAX    D               ;)
40:            MOV     M,A             ;)
41:            MOV     A,C             ;)
42:            STAX    D               ;)DONE ONE DATA EXCHANGE PROCESS
43:            INX     D               ;INCREMENT SORT DATA STORE POINTER
44:            INX     H               ;INCREMENT DATA POINTER
45:            JMP     NEXT2           ;JUMP CHECK DATA END ROUTINE
46:
47:  ;                                *PROGRAM END, RETURN TO CP/M
48:  DONE:
49:            LXI     D,MSGEND        ;)
50:            MVI     C,9             ;)
51:            CALL    BDOS            ;) END MESSAGE OUT SYSTEM CALL
52:
53:            RET                     ;END OF THIS PROGRAM. RETURN TO CP/M
54:
55:  MSGSTRT:  DB      0DH,0AH,'SORT PROGRAM START NOW......',0DH,0AH,'$'
56:  MSGEND:   DB      0DH,0AH,'FUNCTION COMPLETE',0DH,0AH,'$'
57:
58:            END
```

Figure-4.3.2　今からエディットするオリジナル・ファイルの〝SORT．ASM〃のリスト．
ライン No. はリストアウト時に付けたもので，オリジナル・ファイルには付いていない．

```
A>ED SORT.ASM↵ -------ファイル"SORT.ASM"に対して，EDを起動する.

   : *0A↵ ------ディスクからテキストをバッファの半分までロードする．この例ではテキストが短いので，
  1: *4T↵ ------4行分タイプアウト.                                              全部がロードされた.
  1:  ;=======================================================================
  2:  ;               SORT PROGRAM for CP/M Learning System #2
  3:  ;    This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
  4:  ;========================================================= by Y.MURASE ===
  1: *-B-4T↵ ----CPをバッファの最後にセットし，4行前からタイプアウト.
 55:  MSGSTRT: DB      0DH,0AH,'SORT PROGRAM START NOW......',0DH,0AH,'$'
 56:  MSGEND:  DB      0DH,0AH,'FUNCTION COMPLETE',0DH,0AH,'$'
 57:
 58:               END
   : *B↵ ---------CPをバッファの先頭にセット.
  1: *9LT↵ ------CPを9行先の行の頭にセットし，その行をタイプアウト.
 10:  BDOS     EQU     0005H                ;SYSTEM CALL ENTRY POINT
 10: *-4LT↵ -----CPを4行手前の行の頭にセットし，その行をタイプアウト.
  6:               ORG     100H
  6: *8↵ --------CPを8行先の行の頭にセットし，その行をタイプアウト．8LTと同じ.
 14:               CALL    BDOS                ;) START MESSAGE OUT SYSTEM CALL
 14: *-2↵ -------CPを2行手前の行の頭にセットし，その行をタイプアウト．－2LTと同じ.
 12:               LXI     D,MSGSTRT           ;)
 12: *↵ ----------CPを次の行の頭にセットし，その行をタイプアウト．1LTと同じ.
 13:               MVI     C,9                 ;)
 13: *↵ ---------同上.
 14:               CALL    BDOS                ;) START MESSAGE OUT SYSTEM CALL
 14: *20:T↵ -----CPをライン20の頭にセットし，その行をタイプアウト.
 20:               MOV     A,M                 ;GET DATA
 20: *3:T↵ ------CPをライン3の頭にセットし，その行をタイプアウト.
  3:  ;    This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
  3: *21CT↵ -----CPを21文字分進めて，そこから行の終りまでをタイプアウト.
8 bit data at address 4000H to 4FFFH (4K bytes)
  3: *-5CT↵ -----CPを5文字分戻して，そこから行の終りまでをタイプアウト.
SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
  3: *0↵ ------- CPをその行の頭にセットし，その行をタイプアウト.
  3:  ;    This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
  3: *12::14T↵ -------ライン12～14をタイプアウト.
 12:               LXI     D,MSGSTRT           ;)
 13:               MVI     C,9                 ;)
 14:               CALL    BDOS                ;) START MESSAGE OUT SYSTEM CALL
 12: *B↵ -------- CPをバッファの先頭にセット.
  1: *2FSORT^ZT↵ -----CP以後のバッファ内で，2番目の文字列"SORT"を捜し，CPをセットし
8 bit data at address 4000H to 4FFFH (4K bytes)      そこから行の終りまでをタイプアウト.
  3: *0↵ -------- その行の全体をタイプアウト.
  3:  ;    This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
  3: *B↵
  1: *MFSORT^Z3Z0T↵ --------バッファ内の文字列"SORT"をすべて捜し，ラインの頭からそこまでをタイプアウト.
  2:  ;               SORT  タイム・ディレーのZコマンド"3Z"により，1行ずつディレーをとって表示される.
  3:  ;    This program SORT
 22:               JZ      SORT
 22:               JZ      SORT                        ;IF A=C JUMP SORT
 37:  ;                                          *THIS STEP A=C, SORT
 38:  SORT
 43:               INX     D                           ;INCREMENT SORT
 55:  MSGSTRT: DB      0DH,0AH,'SORT
BREAK "#" AT ^Z ----もう見つからないのでブレークがかかった.
 55: *2:T↵ -------CPをライン2の頭にセットし，その行をタイプアウト.
  2:  ;               SORT PROGRAM for CP/M Learning System #2
  2: *19CT↵ ------CPを19文字分進め，残りの行をタイプアウト.
PROGRAM for CP/M Learning System #2
  2: *8D0TT↵ -----CPから8文字分を削除して，ラインの頭からタイプアウト.・"PROGRAM␣"が削除された.
  2:  ;               SORT for CP/M Learning System #2
  2: *12:T↵ ------CPをライン12の頭にセットし，その行をタイプアウト.
 12:               LXI     D,MSGSTRT           ;)
```

Figure-4.3.3 その1

```
   12: *4KT↲ ------CPの行から4行分を削除し，次の行をタイプアウト.
   12:          LXI     D,DATTOP           ;SET TOP ADR IN D,E
   12: *10::13T↲
   10: BDOS     EQU     0005H              ;SYSTEM CALL ENTRY POINT
   11:
   12:          LXI     D,DATTOP           ;SET TOP ADR IN D,E
   13:          LXI     H,DATTOP           ;SET TOP ADR IN H,L
   10: *2:T↲ -------CPをライン2の頭にセットし，その行をタイプアウト
    2: ;                SORT for CP/M Learning System #2
    2: *i↲ ------インサート・モードに入る. iが小文字であることに注目.
    2: ;ABCDEFG hijklnm 12345↲  } 挿入したい行をキーイン. 大文字・小文字が混在していることに注目.
    3: insert INSERT 2 lines↲   }
    4: ^Z  -----インサート・モードを終る.
    4: *1::5T↲ ----- インサート状態の確認.
    1: ;=================================================================
    2: ABCDEFG hijklnm 12345  } インサートされている. 小文字は小文字のままであることに注目.
    3: insert INSERT 2 lines  }
    4: ;                SORT for CP/M Learning System #2
    5: ;   This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
    2: *2:T↲
    2: ABCDEFG hijklnm 12345
    2: *7CIVWXYZ^ZOLT↲ ------ CPを7文字分進めて，文字列"VWXYZ"を挿入し，行の頭からタイプアウト.
    2: ABCDEFGVWXYZ hijklnm 12345 挿入されている.                      Ctrl-Zに注目.
    2: *7CIopqrstu↲ ---------今回はIが大文字であること，"opqrstu"が小文字であること，
*1::7T↲                                          最後にCtrl-Zがないことに注目.
    1: ;=================================================================
    2: ABCDEFGOPQRSTU         } 2行に分かれていることと，挿入されたものが大文字に変換されていることに注目.
    3: VWXYZ hijklmn 12345    }
    4: insert INSERT 2 lines
    5: ;                SORT for CP/M Learning System #2
    6: ;   This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
    7: ;====================================================== by Y.MURASE ===
    1: *7X↲ ------CPから7行を"X$$$$$$$.LIB"ファイルにセーブ.
    1: *-Br↲ -----バッファ内テキストの最後から，上記のファイルを挿入する.
     : *-11T↲ ---- 接続状態の確認                   Rが小文字であることに注目.
   54: MSGSTRT: DB      0DH,0AH,'SORT PROGRAM START NOW......',0DH,0AH,'$'
   55: MSGEND:  DB      0DH,0AH,'FUNCTION COMPLETE',0DH,0AH,'$'
   56:
   57:          END   ------この行が，今までのテキストの最後. この行の後に挿入されている.
   58: ;=================================================================
   59: ABCDEFGOPQRSTU
   60: VWXYZ hijklmn 12345
   61: insert INSERT 2 lines
   62: ;                SORT for CP/M Learning System #2
   63: ;   This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
   64: ;====================================================== by Y.MURASE ===
     : *RDISKDEF↲ -----ログイン・ディスク上の"DISKDEF.LIB"ファイルをCP以後に挿入. 現在のCPはライン65.
     : *62::70T↲ ------ 接続状態の確認.
   62: ;                SORT FOR CP/M LEARNING SYSTEM #2
   63: ;   THIS PROGRAM SORT 8 BIT DATA AT ADDRESS 4000H TO 4FFFH (4K BYTES)
   64: ;====================================================== by Y.MURASE ===
   65: ;        CP/M 2.0 DISK RE-DEFINITION LIBRARY ここからライブラリ・ファイルが挿入された.
   66: ;
   67: ;        COPYRIGHT (C) 1979
   68: ;        DIGITAL RESEARCH
   69: ;        BOX 579
   70: ;        PACIFIC GROVE, CA
   62: *BFEQU^ZOLT↲ -------CPをバッファの先頭にセットして，1番目の文字列"EQU"を捜し，その行をタイプアウト.
   10: DATTOP   EQU     4000H              ;DATA AREA TOP ADDRESS
   10: *JEQU^Z JUXTAPOSITION ^Z;^ZOLT↲ -----"EQU"の後に"␣JUXTAPOSITION␣"を挿入し，";"までの文字を
   10: DATTOP   EQU JUXTAPOSITION ;DATA AREA TOP ADDRESS          削除し，その行をタイプアウト.
   10: *↲
   11: DATEND   EQU     4FFFH              ;DATA AREA END ADDRESS
   11: *S4FFF^Z8000^ZOTT↲ -------文字列"4FFF"を"8000"に置き替えて，その行をタイプアウト.
   11: DATEND   EQU     8000H              ;DATA AREA END ADDRESS
```

Figure-4.3.3
その2

```
    11: *70:T ↲
    70:  ;            PACIFIC GROVE, CA
    70: *#K ↲ -------ライン70から後の行をすべて削除する. #は65535と等価.
     : *BMSEQU^ZDW^ZOTT ↲ -------バッファ内の文字列 "EQU" をすべて "DW" に置き替え, その行をタイプアウト.
    10:  DATTOP  DW JUXTAPOSITION ;DATA AREA TOP ADDRESS
    11:  DATEND  DW       8000H            ;DATA AREA END ADDRESS
    12:  ENDADR  DW       (DATEND+1) SHR 8    ;HIGH 8 BIT OF NEXT '4FFF' = '50'
    13:  BDOS    DW       0005H            ;SYSTEM CALL ENTRY POINT

BREAK "#" AT ^Z -------もう見つからないのでブレークがかかった.
    13: *11::12K ↲------ライン11～12を削除.
    11: *10::12T ↲------削除の確認.
    10:  DATTOP  DW JUXTAPOSITION ;DATA AREA TOP ADDRESS
    11:  BDOS    DW       0005H            ;SYSTEM CALL ENTRY POINT
    12:
    10: *U ↲ ------- 以後, 入力された文字はすべて大文字に変換される.
    10: *i ↲ ------- 小文字でインサート・モードに入る.
    10:  abcdefghijklmn ↲ 小文字を挿入.
    11:  ^Z ------- インサートを終る.
    11: *-  ------- 1行前をタイプアウト.
    10:  ABCDEFGHIJKLMN このように小文字の i コマンドにもかかわらず, 大文字に変換されている.
    10: *-U ↲ ------ Uをキャンセル.
    10: *OV ↲ ------ バッファの空エリア/全体のサイズの表示.
25502/27575   10進のバイト数である.
    10: *-V ↲ ------ ラインNo.の削除.
*↲
BDOS     DW       0005H            ;SYSTEM CALL ENTRY POINT ラインNo.が付いていない.
*V ↲ --------ラインNo.の表示モードにセット.
    11: *↲
    12:                                                            ┐
    12: *↲                                                         │
    13:          LXI     D,DATTOP        ;SET TOP ADR IN D,E       │ ラインNo.付.
    13: *↲                                                         │
    14:          LXI     H,DATTOP        ;SET TOP ADR IN H,L       ┘
    14: *H ↲ -------一旦, 正常にEDを終り, 再びEDに入る. バッファは空になる. オリジナル・ファイルは更新されている.
     : *OA ↲ ------エディットするには再度アペンドが必要.
    1: *3T ↲
    1:  ;=================================================================
    2:  ABCDEFGOPQRSTU
    3:  VWXYZ hijklmn 12345
    1: *O ↲ -------今までの作業のすべてをキャンセル, リセットし, 再びEDが起動した時点に戻る.
                   (先程のHコマンドで, オリジナル・ファイルはすでに更新されている)
O-(Y/N)?Y  --------Oの実行.
     : *OA ↲ ------バッファは空になっているので, 再度アペンドが必要.
    1: *3T ↲
    1:  ;=================================================================
    2:  ABCDEFGOPQRSTU
    3:  VWXYZ hijklmn 12345
    1: *Q ↲ -------すべてを中止し, CP/Mに戻る.

Q-(Y/N)?Y  --------Qの実行.

 A>  -------CP/Mに戻った.
```

Figure-4.3.3 既存ファイルのエディット

エディット作業を終え，最後にEコマンドで正常にEDを終了すると，ED起動前のオリジナル・ファイルは，エクステンションを〝BAK〟とリネームされて，バックアップ・ファイルとして，そのまま保存されている．

Figure-4.3.3のMFやMSコマンドの実行にも2～3表示されているように，EDは，指定されたコマンドが完全に実行できなかった場合，

例えば　BREAK "#" AT^Z

エラー種別　ブレークがかかった時のコマンド

のような，ブレーク・メッセージを出力します．

ブレーク・メッセージが出力されたラインに関しては，コマンドは実行されておらず，もとのままの内容を保っています．

ブレーク・メッセージには，次の種類があります．

? コマンドが間違っている．

> エディット・バッファがいっぱいになり，これ以上追加することができない．

\# 指定回数分のコマンドの実行ができない．バッファあるいはファイルの最終まで来てしまった．

O Rコマンドで読み出すファイルがOPENできない．該当LIBファイルがない．

次に，先のリストで実習されていない"N"と"W"コマンドについて実習します．

実習3 Nコマンド

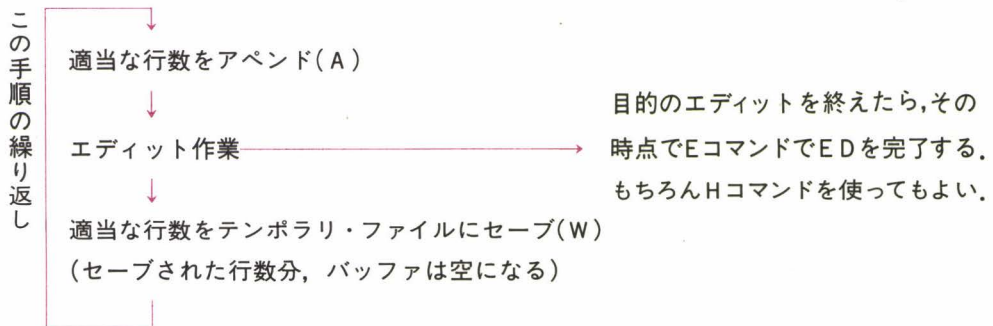
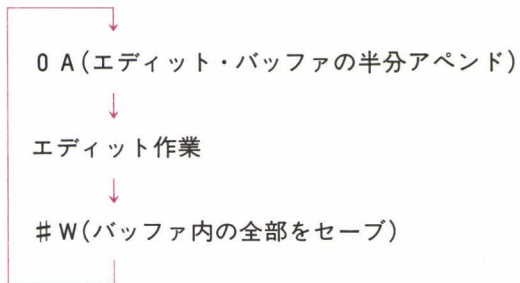
"N"コマンドは，文字列サーチの"F"コマンドと似ていますが，"F"コマンドはエディット・バッファ内のテキストだけに効力があるのに対し，"N"コマンドはアペンドされていないディスク上のオリジナル・ファイルに対しても自動的にアペンドおよびセーブを行いながら，"文字列"をサーチします．"N"コマンドは，ファイルの最後までを対象とするグローバルな"F"コマンドと言えます．

```
A>ED SORT.ASM ↲
    : *NBDOS^ZOTT ↲   Aコマンドを実行せずに，Nコマンドを使っていることに注目．
  10:  BDOS     EQU     0005H              ;SYSTEM CALL ENTRY POINT
  10: *          .
                 .
                 .
```

Figure-4.3.4　ファイル"SORT．ASM"の最初に現れる文字列"BDOS"を捜してタイプアウトする．見つかるまでは，どんなに長大なファイルでもファイルの最後までを自動的にサーチする．

実習4　Wコマンド

一度にエディット・バッファにアペンド（ロード）できるテキストの容量は，48K CP／Mで約27Kバイト，58K CP／Mで約35Kバイトと限られています．よって，これ以上に長いファイルの各部をエディットしたい場合は，テキストをいくつかに分けてエディット・バッファにロードする必要があります．

そのような場合のために，〝W〟コマンドが用意されています．

上記の手順でどんなに長いファイルもエディットが可能です．実際には，

のように行うと，分り易くて便利です．

次にファイル容量が76Kバイトである長いソース・ファイル〝5KBASIC．ASM〟を使って，Wコマンドの実例を示します．

```
A>ED 5KBASIC.ASM ↲ -----容量76Kの長いソース・ファイルに対してEDを起動する.
     : *0A ↲ ------エディット・バッファの半分にテキストをアペンドする.
    1: *5T ↲ ------5行分をタイプアウト. ただの確認.
    1:  ;   HAMPSHIRE COLLEGE 5K BASIC
    2:  ;   ==========================
    3:  ;
    4:  ;   THIS IS VERSION Z1.0 FROM JEFF ZURKOW, WITH THE FOLLOWING
    5:  ;   ADDITIONAL FEATURES:
    1: *0V↲ -----空バッファ/バッファ全体の表示.
13785/27575 ----- 0Aコマンドにより, ちょうど半分だけテキストがロードされている.
         .
  <  Your edit operation  >
         .
     : *#W↲ -----バッファ内のすべての行（#は65535の代用記号）をテンポラリ・ファイルにセーブする. バッファは空になる.
     : *0A↲ ------1回目のアペンドの続きのテキストを, バッファの半分にアペンドする.
  730: *5T↲ -----ただの確認. ラインNo.が730と進んでいることに注目.
  730:          MOV     E,M       ;LOW ORDER TEXT ADDR
  731:          INX     H
  732:          MOV     D,M       ;HIGH ORDER TEXT ADDR
  733:          INX     H         ;ADDR OF PREVIOUS CONTROL STACK ENTRY
  734:          SHLD    CSTKA
         .
  <  Your edit operation  >
         .
     : *#W ↲ ------エディット済みのバッファのすべての行（2回目のアペンドの）をテンポラリ・ファイルにセーブする.
     : *0A ↲ ------空になったバッファに, 続きのテキストをアペンド.
 1473: *5T ↲ ------ただの確認. ラインNo.がさらに進んでいる.
 1473:  LNUMB:  DB      'LINE NUMBER "'
 1474:  NGSQR:  DB      'NEGATIVE SQUARE ROOT "'
 1475:  BOUND:  DB      'BOUNDS "'
 1476:  RDERR:  DB      'READ "'
 1477:  STOVL:  DB      'STORAGE OVERFLOW "'
         .
  < Your edit operation  >
         .
     : *#W ↲
     : *0A ↲
         .
  <  Your edit operation  >   A→W→A→W…の繰り返し. 必要な部分のエディットが終わったら,
         .                    いつでもEコマンドでEDを終了できる.
     : *#W ↲
     : *0A ↲
         .
  < Your edit operation  >
         .
     : *#W↲
     : *0A↲ --------ソース・ファイル・テキストの最後の部分のアペンド.
 4080: *-B-5T ↲ ----テキストの最後の部分をタイプアウト.
 4618:  BOFA:   DS      2         ;START OF FILE ADDRESS
 4619:  MEMTOP: DS      2         ;STORAGE FOR LAST ASSIGNED MEMORY LOCATION
 4620:  ;
 4621:  ;
 4622:          END
         .
  <  Your edit operation  >
         .
     : *E↲ ------ ソース・ファイルのすべての部分のエディットを終えたのでEDを終了する.
A>
```

Figure-4.3.5　エディット・バッファより大きなファイルの編集作業

```
A>STAT 5KBASIC.* ↲

 Recs  Bytes  Ext Acc
  606    76k     5 R/W A:5KBASIC.ASM
  606    76k     5 R/W A:5KBASIC.BAK
Bytes Remaining On A: 17k

A>
```

Figure-4.3.6
ED 終了後のファイルの確認．
正常に ED を終了したので，バックアップ・ファイルが作られている．

解説　ED

ここで，ED 作業における，各時点のディスク上のファイルとエディット・バッファの関係を図示しておきます．作業の通常の流れは 1）→2）→… 6）の順ですが，テキスト・ファイルが短ければ 3）や 4）は行う必要はありません．

エディット作業時のアドバイス

エディット作業は，適当な時点で，H コマンドを使って，今までに出来上っているエディット・バッファの内容を，一旦，ディスク上のオリジナル・ファイルに組み込み，それから新たに作業を再開することをお勧めします．また，E コマンドで，完全に ED を終了してから，"BAK" ファイルを別の名にリネームし，再度 ED に入る方法もあります．いずれも不慮の事故・過失による影響を，少しでも救済するための手段です．

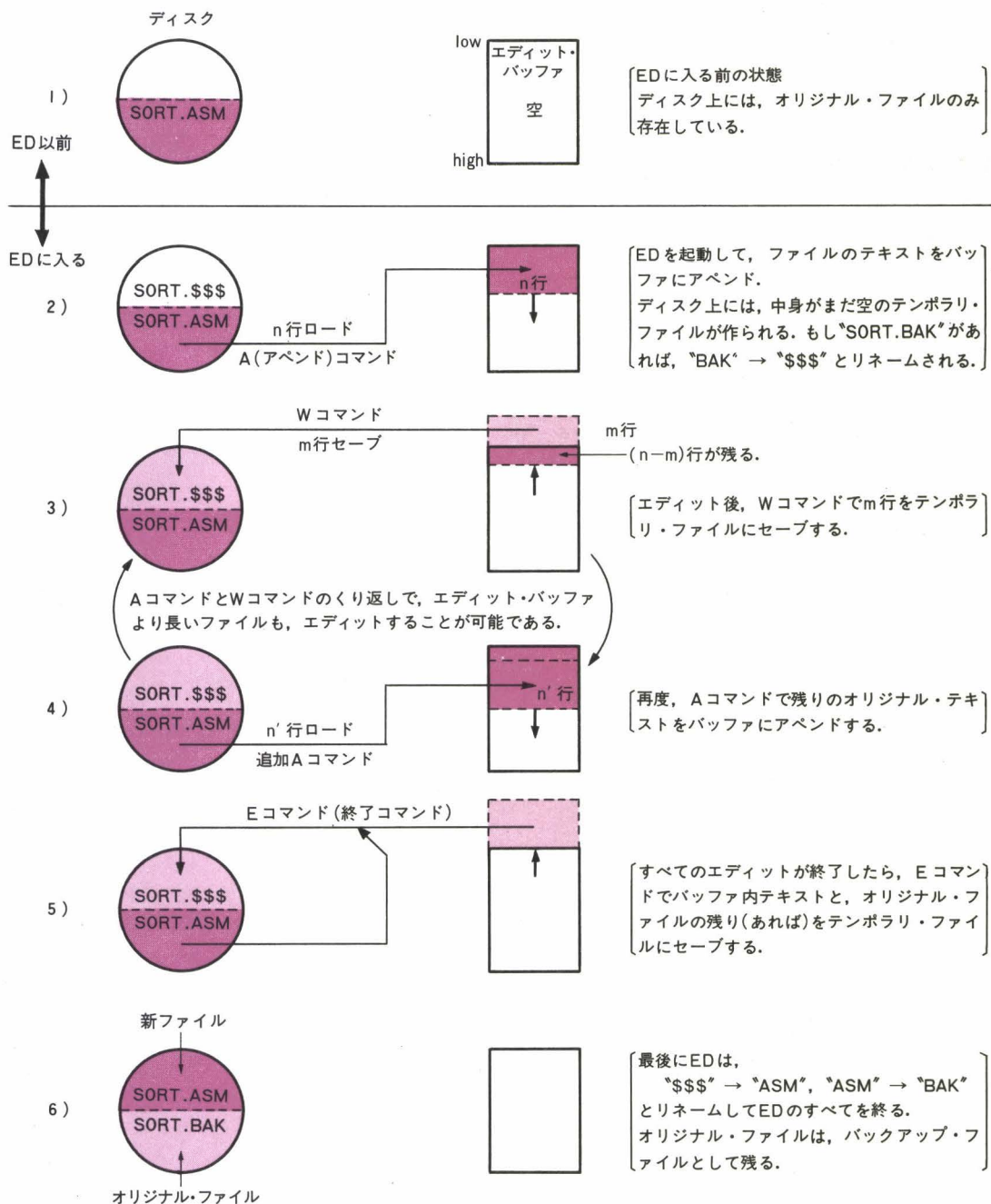



Figure-4.3.7 ED作業の各時点におけるディスク上のファイルとエディット・バッファの関係

4.4 ASM (8080アセンブラ)

——ASseMbler——

コマンド形式・機能

1 ASM␣x : filename

ドライブ x ：上のソース・ファイル〝filename.ASM〟をアセンブルし，生成されたHEXおよびPRNファイルを同じディスク上にセーブする．

2 ASM␣filename.shp

〝s〟で指定されるドライブ上のファイル〝filename.ASM〟をアセンブルし，〝h〟で指定される装置にHEXファイルを，〝p〟で指定される装置にPRNファイルを出力する．

注） x ：はそれがログイン・ディスクの場合，省略できる．

CP／M のアセンブラは，エクステンションが〝ASM〟である8080のアセンブリ・ソース・ファイルをアセンブルし，インテル HEX 形式のオブジェクト・ファイル（エクステンションは〝HEX〟），およびアドレス，オブジェクト・コード，エラー・メッセージなどの付いたプリント形式のファイル（エクステンションは〝PRN〟）を出力します．その様子を次に示します．

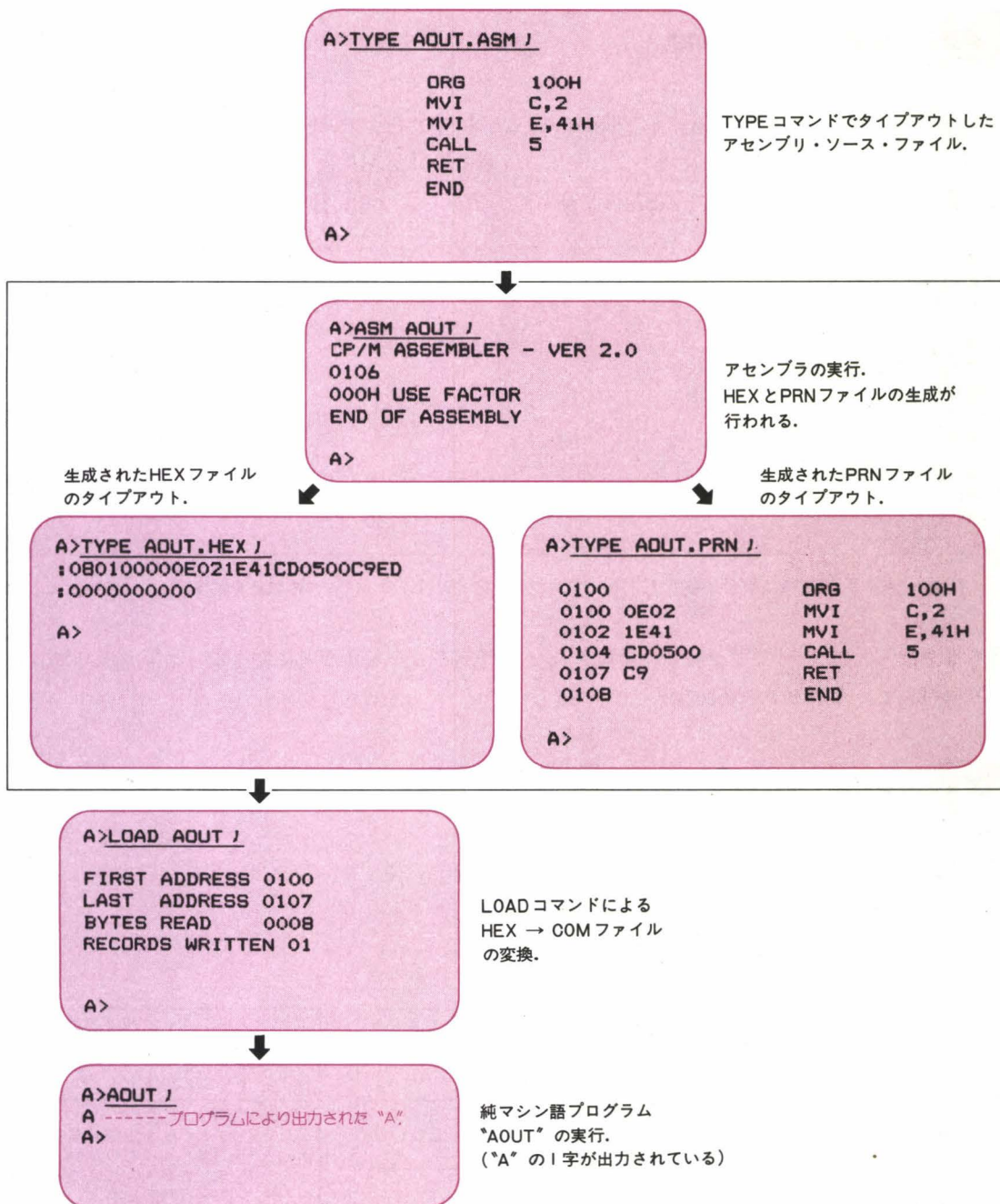



Figure-4.4.1　アセンブリ・ソース・ファイルから即実行可能なコマンド・ファイルができるまでの流れ

実習1 ASM␣x:filename

ドライブx：上のソース・ファイル〝filename．ASM〟をアセンブルし，生成された〝HEX〟と〝PRN〟ファイルを同一ドライブx：上にセーブします．5章の「CP／Mによるマシン語開発実習」で作成するソース・プログラム〝SORT．ASM〟を使って，アセンブラを実行してみましょう．

Figure-4.4.2 アセンブラの実行とアセンブラの実行前・実行後のファイルの状態を示す．

5章では，このソース・ファイルや生成された〝PRN〟ファイルなどの完全なリストを載せて，詳しく解説していますので，ぜひ参照して下さい．

実習2 ASM␣filename.shp

〝shp〟で指示される入出力装置を選択する〝スイッチ〟に従って，アセンブルを行います．スイッチ〝shp〟の機能を次に示します．

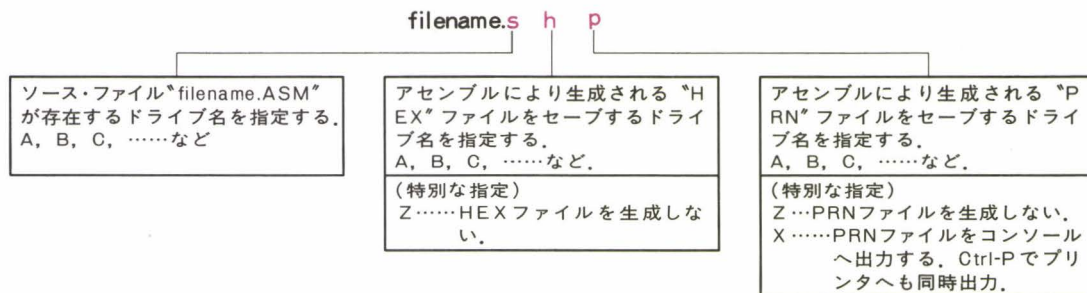



Figure-4.4.3 アセンブラのコマンド書式

コマンド形式	機　　能
A>ASM␣DUMP ⏎	ドライブA：上のソース・ファイル〝DUMP.ASM〞をアセンブルし，生成されたHEXとPRN両ファイルを同一のドライブA：上にセーブする．
B>ASM␣A：DUMP ⏎	ログイン・ディスクのドライブB：上のASMプログラムを起動し，上と同様のことを行う．
B>ASM␣DUMP.AAA ⏎	上とまったく同一の働きをする．
A>ASM␣DUMP.BBA ⏎	ドライブB：上のソース・ファイルをアセンブルし，生成されたHEXファイルをドライブB：上に，PRNファイルをドライブA：上にセーブする．
A>ASM␣DUMP.AZX ⏎	ドライブA：上のソース・ファイルをアセンブルし，HEXファイルは生成せず，PRNファイルのみ生成して，ディスクにはセーブせずに，コンソールに出力する．
A>ASM␣DUMP.AZZ ⏎	ドライブA：上のソース・ファイルをアセンブルするけれども，何も生成しない．但し，アセンブル・エラーなどがコンソールに出力されるので，その確認のために，しばしば使われる．

Figure-4.4.4　ソース・ファイル，〝DUMP. ASM〞をアセンブルする場合のコマンド実例

```
A>DIR SORT.* ⏎
A: SORT      ASM        } アセンブル実行前の“SORT”ファイルに関する確認.

A>DIR B:SORT.* ⏎
NO FILE

A>ASM SORT.ABX ⏎ --------アセンブルのスイッチを“ABX”として実行.
CP/M ASSEMBLER - VER 2.0

                  ;==============================================================
                  ;           SORT PROGRAM for CP/M Learning System #2
                  ;   This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
                  ;======================================================= by Y.MURASE ===

 0100                      ORG     100H
 4000 =           DATTOP   EQU     4000H              ;DATA AREA TOP ADDRESS
 4FFF =           DATEND   EQU     4FFFH              ;DATA AREA END ADDRESS
    .
    .
    .   このようにスイッチ“ABX”のXにより，PRNファイルは直接コンソールに出力される.
    .
    .
 0131 0E09                 MVI     C,9                ;)
 0133 CD0500               CALL    BDOS               ;) END MESSAGE OUT SYSTEM CALL

 0136 C9                   RET                        ;END OF THIS PROGRAM: RETURN TO CP/M

 0137 0D0A534F52MSGSTRT: DB     0DH,0AH,'SORT PROGRAM START NOW......',0DH,0AH,'$'
 0158 0D0A46554EMSGEND:  DB     0DH,0AH,'FUNCTION COMPLETE',0DH,0AH,'$'

 016E                      END

016E
000H USE FACTOR
END OF ASSEMBLY --------アセンブルの終了.
```

```
A>DIR SORT.* ↲
A: SORT     ASM

A>DIR B:SORT.* ↲
B: SORT     HEX--------スイッチ "ABX" のBにより，HEXファイルはドライブB：上に生成されている．
A>
```

Figure-4.4.5 ドライブA：上のソース・ファイル "SORT.ASM" をアセンブルし，生成されたHEXファイルをドライブB：上にセーブし，PRNファイルは，コンソールにのみ出力する実行例．それと，実行前と実行後のファイルの確認．

アセンブリ・ソース・ファイルの書き方

アセンブリ言語のソース・ファイルの書式は，簡単な約束事がいくつかあり，それに準拠してプログラム・ファイルを作成しなければなりません．以下，CP／Mアセンブラのソース・ファイルを作成する上で必要な事柄を，各項目について解説して行きます．

ソース・ファイルのフォーマット（書式）

ここはスペースを置かなくてもよい．

(ラインNo.)	␣ラベル␣	ニーモニック␣	オペランド	;コメント
(通常は書かない)␣	LOOP:␣	MOV ␣	A, B	;get (B) data on (A)

Figure-4.4.6 CP／Mアセンブラのソース・ファイルの書き方

CP／Mアセンブラは「自由フォーマット」であり，書式で決まっているのは，各フィールドの区切り（␣部分）に1つ以上のスペースを置くことと，フィールドの順序のみで，その他は自由に書くことができます．普通は，各フィールドの頭を揃えて読み易くするために␣部にはタブをキーインします．タブは［TAB］キー，またはCtrl-Iでキーインできます．もちろんスペースでもかまいません．

次のリストで，「自由フォーマット」の実例を示しておきます．各ラインの書き方は違っても，アセンブルには関係ないことが分かります．

```
A>TYPE ASMFORM.PRN ↵

 0100                      ORG      100H
 0100 78         LOOP1:    MOV      A,B               ;get (B)data on (A)
 0101 78         LOOP2: MOV A,B;get (B)data on (A)
 0102 78         LOOP3              MOV               A,B               ;get (B)data on (A)
 0103                      END

A>
```

同じ内容のものを，3通りの異なった書き方をしているが，いずれも正しくアセンブルされている．

Figure-4.4.7 CP／Mアセンブラの書式は自由です．

では，それぞれのフィールドについて解説しましょう．

ライン No. について

通常はライン No. を用いません．CP／Mのエディタ以外のエディタの中には，作成したファイルに強制的にライン No. が付いてしまうものもあり，また他のアセンブラではライン No. を必要とするものもあるため，ライン No. 付きのソース・ファイルでも，アセンブルが可能になっています．CP／M アセンブラは，ラインの先頭にある数字を無視します．

ラベルについて

ラベルは，1～16文字の英・数文字を用い，識別は16文字まで行われます．小文字を書くこともできますが，認識は大文字として行われます．また，カナは，ラベルには使用できません（コメントには使用可）．

ラベルの中に読み易すくするために〝$〟記号を任意に挿入することができます．この〝$〟記号をアセンブラは無視します．

ラベルの最後にコロン〝：〟を置くことができます．これは付けておく方が良いでしょう．他のアセンブラでは，〝：〟が必要なものもあるし，何よりもエディタを使う場合，〝：〟が付いているとラベルの文字列サーチが非常に楽になります（同じ〝文字列〟の場合，ラベルと他のものとの区別ができます）．アセンブラは，ラベルの〝：〟を無視します．

★禁止事項

○最初の文字に数字は使えません．

○〝MOV〟，〝LXI〟，…などの8080インストラクション・ニーモニックと同一綴りのものは使えません．

○〝ORG〟，〝DB〟，…などの疑似インストラクションと同一のものも使えません．

○〝A〟，〝B〟，〝M〟，〝SP〟，…など，8080 CPU のレジスタ名と同一のものは使えません．

ニーモニック，オペランドについて

特別な注意事項はありません．通常の書式をとります．8080ニーモニックそのものの解説は，本書では触れません．

コメントについて

各ラインにおいて，セミコロン〝;〞の後の記述はすべて無視されます．よって〝;〞の後に説明文や覚え書きを書いておくと，ドキュメント性の面で役に立ちます．カナを使ってもかまいません．

ラインとラインの間について

ソース・ファイルを読み易くするために，ラインとラインの間をキャリッジ・リターンのみで行間を空けることができます．もちろん，〝;〞をラインの頭に書いてリターンしておくのもよいでしょう．

疑似インストラクション，算術・論理演算子について

これらについては実際にソース・ファイルの中に記述してみて，それをアセンブルした結果をご覧下さい．演算は16ビットで行われますが，その値は，それが使われるオペレーション・コードに合致していなくてはなりません．

```
A>TYPE ASMSORCE.PRN ↲
```

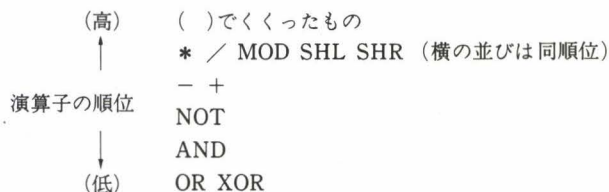
このリストを見る時は，それぞれの結果が現われている左端のアセンブルによるデータと対比して下さい．

```
                   ;====== CP/M ASSEMBLY LANGUAGE SOURCE FILE SAMPLE ======
                   ;                                          by Y.MURASE

0100                        ORG     100H          ;
0100 78            LOOP:    MOV     A,B           ;get  (B)data on  (A)
0101 54            LOOP1    MOV     D,H           ;

0102 78            LOOP2: MOV A,B;get  (B)data on  (A)

0005 =             BDOS     EQU     5             ;
0103 CD0500                 CALL    BDOS          ;

                   FDOS     SET     2000H         ;
0106 CD0020                 CALL    FDOS          ;
                   FDOS     SET     4000H         ;
0109 CD0040                 CALL    FDOS          ;

FFFF =             TRUE     EQU     0FFFFH        ;
0000 =             FALSE    EQU     NOT TRUE      ;
0000 =             I8080    EQU     FALSE         ;
FFFF =             Z80      EQU     TRUE          ;

                            IF      I8080         ;
                            JMP     1234H         ;
                            ENDIF                 ;

                            IF      Z80           ;
010C C37856                 JMP     5678H         ;
                            ENDIF                 ;
```

注記：
- 結果は同じ．（0100 78 と 0102 78）
- ラベルのコロンは自由．（LOOP: と LOOP1）
- 書式は自由．各フィールドをスペースで区切ればよい．
- EQUの使い方．
- SETの使い方．値を変えることが可能．
- アセンブルされていない．
- IF～ENDIFの条件付きアセンブル．"TRUE" のみアセンブルされる．

```
000D =          CR        EQU    0DH                    ;
000A =          LF        EQU    0AH                    ;
010F 58590D0A24           DB     'XY',CR,LF,24H         ;
0114 05007F00             DW     BDOS,7FH               ;
0118                      DS     32                     ;
0138 =          KOKO      EQU    $                      ;
0138 C33801               JMP    KOKO                   ;

000A =          PARM1     EQU    0000$1010B             ;
000A =          PARM2     EQU    12Q                    ;
000A =          PARM3     EQU    12O                    ;
000A =          PARM4     EQU    10                     ;
000A =          PARM5     EQU    10D                    ;
000A =          PARM6     EQU    0AH                    ;

4000 =          AA        EQU    4000H                  ;
0003 =          BB        EQU    3                      ;

4003 =          X1        EQU    AA+BB                  ;
3FFD =          X2        EQU    AA-BB                  ;
C000 =          X3        EQU    AA*BB                  ;
1555 =          X4        EQU    AA/BB                  ;
0001 =          X5        EQU    AA MOD BB              ;
BFFF =          X6        EQU    NOT AA                 ;
0000 =          X7        EQU    AA AND BB              ;
4003 =          X8        EQU    AA OR BB               ;
4003 =          X9        EQU    AA XOR BB              ;
0018 =          X10       EQU    BB SHL BB              ;
0800 =          X11       EQU    AA SHR BB              ;

007F =          CPM$ASM   EQU    0111$1111B             ;
013B 3E7F                 MVI    A,CPMASM               ;

013D C5D5E5               PUSH B! PUSH D! PUSH H!       ;
0140 E1D1C1               POP H! POP D! POP B           ;

0143 C34701     ABCDEFGHIJKLMNO1: JMP ABCDEFGHIJKLMNO2  ;
0146 00                           NOP                   ;
0147 C34301     ABCDEFGHIJKLMNO2: JMP abcdefgHIJKLMNO1  ;

014A                      END                           ;

A>
```


Figure-4.4.8　疑似インストラクション，算術・論理演算子の実例.

（高）	（　）でくくったもの
↑	＊ ／ MOD SHL SHR（横の並びは同順位）
	− ＋
演算子の順位	NOT
↓	AND
（低）	OR XOR

ソース・プログラムのアセンブル・エラーについて

アセンブル時，ソース・プログラムに文法上，記述上などの各種のエラーが発見された場合，アセンブラはそのつどコンソールにエラー種別を示す1文字のエラー・メッセージと，そのエラーが存在するラインをタイプアウトします．

故意にエラーのあるソース・プログラム（ASMER．ASM）を作り，実際にアセンブルした結果を次に示します．

```
A>ASM ASMER ↲

CP/M ASSEMBLER - VER 2.0
E                     MOV   A B
E0100 40              NOP
N               MAC   MACRO
V0101 3E80            MVI   A,80H*3
P0103 00        PHASE: NOP
O0105 424E4D2C2E      DB    'QWERTYUIOPZXCVBNMQWERTYUIOP . . . . . ZXCVBNM'
R010B 33              INX   A
U010C C30000          JMP   UNDEF
L               MISS1 MISS2 H
010F
000H USE FACTOR
END OF ASSEMBLY

A>
```


Figure-4.4.9　アセンブラ実行時コンソールに出力されるエラー・メッセージの例．

```
 0100                 ORG   100H
E                     MOV   A B
E0100 40              NOP
N               MAC   MACRO
V0101 3E80            MVI   A,80H*3
P0103 00        PHASE: NOP
 0104 00        PHASE: NOP
O0105 424E4D2C2E      DB    'QWERTYUIOPZXCVBNMQWERTYUIOP . . . . . ZXCVBNM'
R010B 33              INX   A
U010C C30000          JMP   UNDEF
L               MISS1 MISS2 H
 010F                 END
```


Figure-4.4.10　生成された PRN ファイルにも，エラー・メッセージが付いています．

次に，各種エラー・メッセージとそれらの主な原因をまとめて示します．

エラーメッセージ	エラーの主な原因
D	Dataエラー．エクスプレッションの値がデータ・エリアに適合しない．
E	Expressionエラー．オペレーション・コードのミスタイプや演算子の使い方を誤った場合などの表現上のエラー
L	Labelエラー．ラベルを正しく用いない場合．
N	not-implementedエラー．MACなどの上位アセンブラにはあるが，CP/Mアセンブラにはインプリメントされていない特別な命令を用いた場合など．
O	Overflowエラー．エクスプレッションの値が，その場合の限度を越えた時など．
P	Phaseエラー．アセンブラ実行中の各パスで，ラベルの値が異なった場合．
R	Registerエラー．オペレーションに対して，適合しないレジスタが指定された場合．
U	Undefinedラベル・エラー．オペランドで使用されたラベルが，どこにも定義されていない場合．
V	Valueエラー．オペランドや，エクスプレッションの値が，それに対する範囲外である場合．

（注．デジタルリサーチのCP/Mマニュアルには上記の種類しか記載されていないが，実際にはこのほかのエラー・メッセージも用意されている．）

Figure-4.4.11　エラー・メッセージとエラーの主な原因

4.5　LOAD　(HEXファイル➡COMファイル変換プログラム)

――LOAD program――

コマンド形式・機能

LOAD␣x : filename

ドライブx：上のインテルHEX形式のファイル，"filename.HEX" から，即実行可能な純マシン語の "COM" ファイルを生成し，同一ディスク上にセーブする．

注）　x：は，それがログイン・ディスクの場合，省略できる．

実習　LOAD␣x:filename

```
A>LOAD 5KBASIC ↲

FIRST ADDRESS 0100
LAST  ADDRESS 1E05
BYTES READ    1C35
RECORDS WRITTEN 3B

A>
```

Figure-4.5.1
ログイン・ディスク上の "5 KBASIC．HEX" からCOMファイルを生成し，同じディスクにセーブする．

```
A>STAT 5KBASIC.* ↲

 Recs  Bytes  Ext Acc
   59     8K    1 R/W A:5KBASIC.COM
Bytes Remaining On A: 84K

A>
```

Figure-4.5.2
セーブされたCOMファイルの確認．
この純マシン・オブジェクト・ファイルは，以後，ファイル名だけをキーインすることにより，トランジェント・コマンドとして実行される．
(BASICの名前と，実際のファイル長とは関係ありません)

```
A>5KBASIC↲

BASIC/5 INTERACTIVE INTERPRETER    V Z1.0  10/16/77

NEW OR OLD? NEW↲
NEW PROGRAM NAME: TEST↲

READY
10 A=5↲
20 B=63↲
30 C=88↲
40 PRINT "SC = ",A*B/C↲
50 END↲
RUN↲

SC =  3.57954

READY
  .
  .
  .
  .
```

Figure-4.5.3　LOAD コマンドによって生成・セーブされた，"5KBASIC．COM" を実行してみる．BASIC言語が起動・実行されている．

解説 LOAD

LOAD コマンドは，その名前からは SAVE コマンドに相対する機能のように思えますが，全く違って，HEX ファイル⇨COM ファイルの変換プログラムですのでご注意下さい．

HEX ファイルのメモリ上へのロード・アドレス（ソース・プログラムの ORG で指定されているアドレス）が，100H未満である場合 LOAD コマンドは実行されず，エラー・メッセージが出力されます．LOAD コマンドによって生成される "COM" ファイルはアドレス100Hからのメモリ上にロードされた後，必ず100Hスタートで実行されるので，100H未満の ORG アドレスを持つ HEX ファイルを LOAD コマンドで処理することはあり得ないのです．ORG が 0 Hである HEX ファイル(PTBOOT.HEX――CP／Mシステムのブート・プログラム）に，LOAD コマンドを実行した例を示します．

```
A>LOAD PTBOOT32↲

ERROR: INVERTED LOAD ADDRESS, LOAD ADDRESS 0000
A>
```

Figure-4.5.4　実行開始アドレスが0Hであるプログラムの HEX ファイルに対して，LOAD コマンドを実行した例.
100H未満のロード・アドレスを持つ HEX ファイルは LOAD できない.

また100H以上の ORG アドレスを持つ HEX ファイル——例えば ORG 2000H などは LOAD 可能ですが，生成される COM ファイルは，ORG アドレスの2000Hからセーブされるのではなく，不用である100H～1FFFH までも一緒にセーブされます．しかもこの COM ファイルをトランジェント・コマンドとして実行することはできません．トランジェント・コマンドは，前述の通り，メモリにロードされた後は，やはり100Hスタートで実行されるからです．もちろんアドレス100Hに，アドレス2000Hへのジャンプ命令を入れておけば実行は可能になります．

また，上記のような，特別なロード・アドレスの HEX ファイルを，純マシン語に変換して，メモリ上にロードするには，次ページの DDT コマンドで行うことができます．

4.6　DDT　(8080デバッガ)

――Dynamic Debugging Tool――

コマンド形式・機能

1　**DDT**

DDT プログラムを起動し，DDT 内の全コマンドの実行を可能にする．

2　**DDT␣x : filename.ext**

DDT を起動すると同時に，ドライブ x：上のファイル，〝filename.ext〃 をアドレス100H からメモリにロードする．HEX ファイルの場合はそのロードアドレスに従う．

注1)　x：は，それがログイン・ディスクの場合，省略できる．

注2)　DDT も ED と同じく，コントロール・キーによるライン・エディッティング機能が使えます（「実習のはじめに」参照）．

DDT内コマンドの機能

アセンブル

Assss　1ステップ毎のライン・アセンブルをし，そのオブジェクト・コードをアドレス ssssH からロードして行く．
オペランドに書く数値はすべて HEX であるが，〝H〃 を付けてはいけない．
終了するにはスペースを入力してリターンする．

ダンプ

Dssss, eeee
アドレス ssssH から eeeeH までのメモリ内容をダンプする．16進表示の右側には，それに対応するアスキー・キャラクタが表示される．アスキー・キャラクタ以外のコードは〝.〃で示される．カナも表示されるように DDT の一部を変更したものもある．

D, eeee　現在のディスプレイ・アドレス（前回のDコマンドが表示した最終アドレスの次）から，eeeeH までをダンプする．上記のコマンドの ssss を省略したもの．

Dssss　アドレス ssssH から12行をダンプする（1行は16バイト）．

D　現在のディスプレイ・アドレスから12行をダンプする．

フィル

Fssss, eeee, cc
アドレス ssssH から eeeeH までのメモリ全部に，データ ccH を書き込む．

ゴー（実行）

Gssss, bbbb

Gssss, bbbb, b'b'b'b'

1つまたは2つのブレーク・ポイント（bbbbHとb'b'b'b' H）を設けて，アドレス ssss Hから，リアルタイムで被テスト・プログラムを実行する．

G, bbbb

G, bbbb, b'b'b'b'

スタート・アドレスが現在のプログラム・カウンタ（Xコマンドで表示，XP コマンドで変更可能）であること以外は，上記と同様．

Gssss ブレーク・ポイントを設けずに，アドレス ssssH からリアルタイムで実行する．

G ブレーク・ポイントを設けずに，現在のプログラム・カウンタからリアルタイムで実行する．

ヘキサ（計算）

Hxxxx, yyyy

16進の数値 xxxx と yyyy の和（xxxx＋yyyy）と差（xxxx－yyyy）を，16進で表示する．

インプット

Ifilename . ext

アドレス 5CH の FCB（ファイル・コントロール・ブロック）に，任意のファイル名〝filename . ext〟をセットする．但し，ドライブ名を付けることはできない．通常は次のRコマンドと共に，任意のファイルをメモリにロードする場合に使われる．

リード

R 現在の FCB にセットされているファイル（通常は上記 I コマンドによってセットされる）を，それが HEX ファイルの場合は，それ自身が持つアドレスに純マシン・コードに変換してロードする．HEX ファイル以外は，アドレス100Hからのメモリにそのままの形でロードする．

Rbbbb ロード・アドレスが，＋bbbbH分バイアス（オフセット）される以外，上記と同様．バイアス値を加えたロード・アドレスが FFFFH を越えた場合，FFFFH の次は 0 H として数え，0 Hを越えた分がロード・アドレスになる．

リスト（逆アセンブル）

Lssss, eeee

アドレス ssssH から eeeeHまでのメモリ内容を逆アセンブルする．

Lssss アドレス ssssH から11ライン（ステップ）分を逆アセンブルする．

L 現在のリスト・アドレス（前回のLコマンドの最終アドレスの次）から11ライン分を逆アセンブルする．

ムーブ

Mssss, eeee, nnnn

アドレス ssssHから eeeeHまでのメモリの内容を，アドレスnnnnHからへブロック転送する.

セット

Sssss　アドレス ssssH のメモリ内容を表示し，必要があれば任意のデータに変更する.
リターン・キーでアドレスを順次進めて行き，終了にはピリオド〝.〟を入力し，リターンで終る.

トレース

Tn　現在のプログラム・カウンタ（XP コマンドで任意設定可能）から，n ステップ分のトレース実行．各ステップごとの CPU の状態が表示される．n＝1のときは1を省略できる．〝DEL〟キー，または何らかのキー入力で実行のブレークができる.

Un　最終ステップの CPU の状態のみ表示され，中間のステップは表示されない上記 Tn コマンドである.

イグザミン

X　現在の CPU が保持している各レジスタ，カウンタ，フラグ，命令などを表示する.

Xr　r で指定するレジスタ，カウンタ，フラグを任意の値に設定する.
r は，次のいずれかを指定する．C（キャリー），Z（ゼロ），M（マイナス），E（偶数パリティ），I（aux キャリー），A, BC, DE, HL（各レジスタ），S（スタック・ポインタ），P（プログラム・カウンタ）

実習1　DDT
実習2　DDT␣x:filename.ext

DDT の実習には，サンプル・プログラムとして，DDT とは密接な関係にある第5章の「CP／M によるマシン語開発実習」で作成される，ソート・プログラムを用いますので，5章を必ず参照して下さい．

```
A>DDT↲
DDT VERS 2.2
-ISORT.HEX↲ --- Iコマンド.
-R↲------------ バイアスを付けないRコマンド.
NEXT  PC
016E 0000------下と同じ結果.
-
```

Figure-4.6.1
まず DDT のみを起動しておき，DDT 内コマンドのIとRで "SORT. HEX" をメモリにロードする．この場合ファイル形式が何であっても，Rコマンドにバイアスを付けることにより，メモリ上の任意のアドレスにロード可能である．バイアスを付けない場合，HEX ファイル以外は100Hに固定．但し，Iコマンドにはドライブ名を指定する機能がないので，ログイン・ディスク上のファイルしかロードできない（別の方法で可能）．

‖　　結果は同じ．

```
A>DDT SORT.HEX↲
DDT VERS 2.2
NEXT  PC
016E 0000 ----- 上と同じ結果.
-
```

Figure-4.6.2
DDT の起動と同時に "SORT. HEX" をメモリにロードする．この場合，ロードアドレスは HEX ファイル以外は100Hに固定となる．ファイル名にドライブ名 x：を付けることにより，任意のドライブからのロードが可能である．

注）　HEX ファイルのロードは，1，2いずれの場合もファイル自身が持つロード・アドレスに従って純マシン・コードに変換されてロードされます．さらに，Rコマンドでバイアスを付けて任意のアドレスにロードすることも可能です．

実習3　DDT内全コマンド

DDT 内のすべてのコマンドの様々な使い方を実習します．ほとんどすべての使い方を示していますので，良い参考になると思います．キー入力とそれに対する DDT の応答をよく見て下さい．また5章でもソート・プログラムのデバッグを行いながら，必要なコマンドの実例を示していますので，必ず参照して下さい．

```
A>DDT SORT.HEX    -----DDTを起動し, "SORT.HEX"をメモリにロードする.
DDT VERS 2.2
NEXT  PC
016E 0000   --------ロードされたオブジェクトの最終アドレスは(16E-1)Hである.
-D100,12F ↵ --------100H～12FH間のダンプ.
0100 11 37 01 0E 09 CD 05 00 11 00 40 21 00 40 0E 00 .7........@!.@..
0110 7E B9 CA 25 01 23 7C FE 50 C2 10 01 0C CA 2E 01 ~..%.#|.P.......
0120 62 6B C3 10 01 1A 77 79 12 13 23 C3 16 01 11 58 bk....wy..#....X
-D,14F ↵ ------- 続きから14FHまでのダンプ.
0130 01 0E 09 CD 05 00 C9 0D 0A 53 4F 52 54 20 50 52 .........SORT PR
0140 4F 47 52 41 4D 20 53 54 41 52 54 20 4E 4F 57 2E OGRAM START NOW.
-D ↵ --------続きから12ライン分のダンプ.
0150 2E 2E 2E 2E 2E 0D 0A 24 0D 0A 46 55 4E 43 54 49 .......$..FUNCTI
0160 4F 4E 20 43 4F 4D 50 4C 45 54 45 0D 0A 24 0B 7B ON COMPLETE..$.{
0170 E6 07 C2 7A 01 E3 7E 23 E3 6F 7D 17 6F D2 83 01 ...z..~#.o}.o...
0180 1A 84 12 13 C3 69 01 D1 2E 00 E9 0E 10 CD 05 00 .....i..........
0190 32 5F 1E C9 21 66 1E 70 2B 71 2A 65 1E EB 0E 11 2_..!f.p+q*e....
01A0 CD 05 00 32 5F 1E C9 11 00 00 0E 12 CD 05 00 32 ...2_..........2
01B0 5F 1E C9 21 68 1E 70 2B 71 2A 67 1E EB 0E 13 CD _..!h.p+q*g.....
01C0 05 00 C9 21 6A 1E 70 2B 71 2A 69 1E EB 0E 14 CD ...!j.p+q*i.....
01D0 05 00 C9 21 6C 1E 70 2B 71 2A 6B 1E EB 0E 15 CD ...!l.p+q*k.....
01E0 05 00 C9 21 6E 1E 70 2B 71 2A 6D 1E EB 0E 16 CD ...!n.p+q*m.....
01F0 05 00 32 5F 1E C9 21 70 1E 70 4A 01 00 9C B4 13 ..2_..!p.pJ.....
0200 C3 83 06 00 00 00 C3 4F 03 C3 24 05 2A 73 1E EB .......O..$.*s..
-F16E,1FF,00 ↵ ------16EH～1FFHの間をデータ "00" でうめる.
-D150 ↵ ------- 結果の確認のダンプ. 150Hから12ライン分のダンプ.
0150 2E 2E 2E 2E 2E 0D 0A 24 0D 0A 46 55 4E 43 54 49 .......$..FUNCTI
0160 4F 4E 20 43 4F 4D 50 4C 45 54 45 0D 0A 24 00 00 ON COMPLETE..$..
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
01F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0200 C3 83 06 00 00 00 C3 4F 03 C3 24 05 2A 73 1E EB .......O..$.*s..
-L100,10F ↵ ------100Hから10FHを逆アセンブル.
  0100  LXI  D,0137
  0103  MVI  C,09
  0105  CALL 0005
  0108  LXI  D,4000
  010B  LXI  H,4000
  010E  MVI  C,00
  0110
-L ↵ ------続きから11ライン分を逆アセンブル.
  0110  MOV  A,M
  0111  CMP  C
  0112  JZ   0125
  0115  INX  H
  0116  MOV  A,H
  0117  CPI  50
  0119  JNZ  0110
  011C  INR  C
  011D  JZ   012E
  0120  MOV  H,D
  0121  MOV  L,E
```

Figure-5.3.3またはFigure-5.4.3のSORTプログラムのソース・リストと対比して下さい.

Figure-4.6.3
その1

```
-LBA00 ↵ -------- BA00Hから11ライン分を逆アセンブル．このエリアは48K CP/MのBIOSの先頭に当る．
  BA00   JMP   BA9C         16K分小さい32K CP/Mのジャンプ・ベクトルの部分が，4章のFigure-4.10.8にあるので参照．
  BA03   JMP   BAF8
  BA06   JMP   BB51
  BA09   JMP   BB5A
  BA0C   JMP   BB8D
  BA0F   JMP   BD59
  BA12   JMP   BD64
  BA15   JMP   BD69
  BA18   JMP   BBF1
  BA1B   JMP   BBB2
  BA1E   JMP   BBF3
-F200,6000,00 ↵ ------- 後の実習のため，200H～6000Hを0クリアしておく．
-A500 ↵ ------- 500Hから，ライン・アセンブルで，オブジェクト・コードをロードする．
0500  MVI C,2 ↵   | ニーモニック，オペランド等を入力すること．
0502  MVI E,5A ↵  | 数値はHEXであるが，Hは付けないこと．
0504  CALL 5 ↵    | RST7はDDTでの実行のブレークとして用いられている．
0507  RST 7 ↵     |
0508  ↵ ---------- 最後はリターンでAコマンドを終了．
-D500,50F ↵ ------- 結果の確認．
0500 0E 02 1E 5A CD 05 00 FF 00 00 00 00 00 00 00 00 ...Z............
-G500 ↵ ------- 500Hからプログラムの実行．先ほど入力したのは，文字“Z”をコンソールに出力する．
Z*0507             システム・コールのプログラムであり，ここに“Z”が出力されている．
-H4000,200 ↵ ------- 2つのHEX数の和と差を求める．
4200 3E00
-S500 ↵ ------- 500Hからのメモリの内容表示と，書き替え．
0500 00 12 ↵  |
0501 00 34 ↵  | それぞれのアドレスの内容が書き替えられて行く．
0502 00 56 ↵  | 左側が元のデータ．
0503 00 78 ↵  |
0504 00 9A ↵  |
0505 00 . ↵ ----- ピリオドでSコマンドを終る．
-D500,50F ↵ ----- 結果の確認．先ほどのライン・アセンブルによるデータが書き替えられている．
0500 12 34 56 78 9A 05 00 FF 00 00 00 00 00 00 00 00 .4VX............
-M100,170,2000 ↵ ------ 100H～170Hのデータを，2000Hからブロック転送する．
-D2000,206F ↵ -------- 結果の確認．本リスト最初のデータと比較して下さい．
2000 11 37 01 0E 09 CD 05 00 11 00 40 21 00 40 0E b0 .7.......@!.@..
2010 7E B9 CA 25 01 23 7C FE 50 C2 10 01 0C CA 2E 01 ~..%.#|.P.......
2020 62 6B C3 10 01 1A 77 79 12 13 23 C3 16 01 11 58 bk....wy..#....X
2030 01 0E 09 CD 05 00 C9 0D 0A 53 4F 52 54 20 50 52 .........SORT PR
2040 4F 47 52 41 4D 20 53 54 41 52 54 20 4E 4F 57 2E OGRAM START NOW.
2050 2E 2E 2E 2E 2E 0D 0A 24 0D 0A 46 55 4E 43 54 49 .......$..FUNCTI
2060 4F 4E 20 43 4F 4D 50 4C 45 54 45 0D 0A 24 00 00 ON COMPLETE..$..
-IDDT.COM ↵ ----- ファイル名“DDT.COM”を5CHからのFCBにセット．
-R3F00 ↵ --------- 3F00H分のバイアスを加えて，上記ファイルをメモリにロードする．
NEXT  PC              結局(3F00H＋100H)からロードされる．
5300 1207
-D3FE0,402F ↵ ------- 結果の確認．4000Hから“DDT.COM”がロードされている．
3FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4000 01 BC 0F C3 3D 01 43 4F 50 59 52 49 47 48 54 20 ....=.COPYRIGHT
4010 28 43 29 20 31 39 38 30 2C 20 44 49 47 49 54 41 (C) 1980, DIGITA
4020 4C 20 52 45 53 45 41 52 43 48 20 20 20 20 20 20 L RESEARCH
-X ↵ ------ 現在のCPUの状態を表示する．物理的にはZ80であっても8080として表示する．
C0Z1M0E1I0 A=00 B=005A D=005A H=0000 S=0100 P=1207 RST  07
-XP ↵ ------ プログラム・カウンタの値を任意に設定する．Pの代りに，C，Z，M，…A,B,…Sなど任意である．
P=1207 100 ↵ ---- 現在のPの値が表示され，続いて任意の値をキーインする．
```

Figure-4.6.3
その2

```
-X↲ ------再度確認．P=100に変っている．
COZ1MOE1IO A=00 B=005A D=005A H=0000 S=0100 P=0100 LXI  D,0137
-T↲ ------現在のプログラム・カウントから1ステップのトレース実行．
COZ1MOE1IO A=00 B=005A D=005A H=0000 S=0100 P=0100 LXI  D,0137*0103
-T↲ ------もう1ステップのトレース実行．表示内容が変化していることに注目．
COZ1MOE1IO A=00 B=005A D=0137 H=0000 S=0100 P=0103 MVI  C,09*0105
-T↲ ------もう1ステップ．
COZ1MOE1IO A=00 B=0009 D=0137 H=0000 S=0100 P=0105 CALL 0005*0005
-G,125↲ ------ブレーク・ポイントを125Hに置いて，リアルタイムで実行．
                ブレーク・ポイントは，もう1つ（合計2箇所）設けることが可能．

SORT PROGRAM START NOW...... テスト・プログラムからのメッセージ出力．
*0125 ------ブレーク・ポイントで止まった．
-T8↲ -------そこから続けて8ステップ分のトレース実行．アキュムレータの値などに注目．プログラム・リスト参照．
COZ1MOE1I1 A=00 B=0000 D=4000 H=403E S=0100 P=0125 LDAX D
COZ1MOE1I1 A=01 B=0000 D=4000 H=403E S=0100 P=0126 MOV  M,A
COZ1MOE1I1 A=01 B=0000 D=4000 H=403E S=0100 P=0127 MOV  A,C
COZ1MOE1I1 A=00 B=0000 D=4000 H=403E S=0100 P=0128 STAX D
COZ1MOE1I1 A=00 B=0000 D=4000 H=403E S=0100 P=0129 INX  D
COZ1MOE1I1 A=00 B=0000 D=4001 H=403E S=0100 P=012A INX  H
COZ1MOE1I1 A=00 B=0000 D=4001 H=403F S=0100 P=012B JMP  0116
COZ1MOE1I1 A=00 B=0000 D=4001 H=403F S=0100 P=0116 MOV  A,H*0117
-U8↲ ------続けてUコマンドによる8ステップ分の実行．最終ステップのみ表示される．
COZ1MOE1I1 A=40 B=0000 D=4001 H=403F S=0100 P=0117 CPI  50*0119
-S136↲
0136 C9 FF↲   Sコマンドで，本プログラムの最終出口の "RET" 命令(C9)を
0137 OD .↲    "RST7" (FF)に書き替えておく．
-G↲ ------先程の続きから，一気にリアルタイムで実行．

FUNCTION COMPLETE テスト・プログラムのエンド・メッセージが出力された．
*0136 -----------テスト・プログラムの最終で "RST7" によるブレークがかかった．
-D4000,4FFF↲ ---データ・エリアのソート状態の確認．
4000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
  .
  .    テスト・プログラムは順調に動作し，この様に00～FFまで，値の小さい順に
  .    並べ替えられていることが分かる．
40B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ................
40C0 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ................
40D0 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ................
40E0 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ................
40F0 01 01 01 01 01 01 01 01 02 02 02 02 02 02 02 02 ................
4100 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
4110 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 ................
4120 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 03 ................
4130 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 ................
  .
  .    第5章「CP/Mによるマシン語開発実習」を必ず参照して下さい．
  .
4FE0 FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE ................
4FF0 FE FE FE FE FE FE FF FF FF FF FF FF FF FF FF FF ................
-G0↲ --------Gコマンドでアドレス0にジャンプさせ，リブートを起こさせて，CP/Mに戻る．

A>   --------Ctrl-Cをキーインしても同じ結果である．
```

Figure-4.6.3　DDT 内のすべてのコマンドの実習．

■こんなテクニック，知っていますか？■

IとRコマンドでは，ログイン・ディスク以外のドライブからファイルをロードすることが出来ません．でもDDTに入ってから，別のドライブからどうしてもロードしたい！さてどうしますか？

Sコマンドで FCB（ファイル・コントロール・ブロック）の先頭アドレスの5CHの1バイトを書き替えるだけで解決できます．

ログイン・ディスクがA：で，DDTを起動後ドライブB：上の〝SORT.COM〟をロードする場合の実例を示します．

```
A>DDT ↲ ------------ログイン・ディスクA：からDDTを起動.
DDT VERS 2.2
-ISORT.COM ↲ ------まず I コマンドでファイル名だけ入力する.
-S5C ↲ --------------Sコマンドでアドレス5CHを目的のドライブNo.に書き替える.
005C 00 02↲          A：=01，B：=02，C：=03，…となる(00はログイン・ディスクを表す).
005D 53 .↲
-R↲ ----------------Rコマンドの実行．ドライブB：上のファイルがロードされる.
NEXT  PC
0180 0100

-G100,136 ↲  -----試しに，ロードされた〝SORT.COM〟を実行してみる．136Hはブレーク・ポイント.

SORT PROGRAM START NOW......
FUNCTION COMPLETE            } このように実行され，ロードが正常に行われていたことが分かる.
*0136
-
      .
      .
```

Figure-4.6.4　IとRコマンドによりログイン・ディスク以外のドライブからファイルをロードするテクニック．

DDT内コマンドのセパレータとして使われるコンマ〝,〟は，スペースや，〝：〟，〝；〟などで代用できます．その一例を次に示します．

```
A>DDT↲

DDT VERS 2.2
-D100,11F↲ ------ダンプ.
0100 01 BC OF C3 3D 01 43 4F 50 59 52 49 47 48 54 20 ....=.COPYRIGHT
0110 28 43 29 20 31 39 38 30 2C 20 44 49 47 49 54 41 (C) 1980, DIGITA

-D100 11F↲ ----スペースを代用.
0100 01 BC OF C3 3D 01 43 4F 50 59 52 49 47 48 54 20 ....=.COPYRIGHT
0110 28 43 29 20 31 39 38 30 2C 20 44 49 47 49 54 41 (C) 1980, DIGITA
```

```
-L0,7 ↵ ---------逆アセンブル.
  0000  JMP  BA03
  0003  NOP
  0004  NOP
  0005  JMP  9C00
  0008

-L0 7↵ ----------スペースを代用.
  0000  JMP  BA03
  0003  NOP
  0004  NOP
  0005  JMP  9C00
  0008

-F100,10F,FF↵ -----フィル.
-D100 11F↵
0100 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0110 28 43 29 20 31 39 38 30 2C 20 44 49 47 49 54 41 (C) 1980, DIGITA

-F110 11F AA↵ -----スペースを代用.
-D100 11F↵
0100 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ................
0110 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA ................
-
   .
   .
   .
```

Figure-4.6.5　セパレータとしてのコンマは，スペース，コロン，セミコロンで代用できる

解説　DDT

DDTはトランジェント・プログラムではありますが，そのプログラムのメモリ上のロードのされ方が，通常のものとは大きく異なります．

これはDDTが〝デバッガ〟であるために，被テスト・プログラムをロードするためのユーザー・エリアを開放しておかねばならず，DDTのプログラム自身は，邪魔にならないようTPAの最後部へ，しかもCCP部へも侵入して置かれるのです．

そのDDT起動の流れをFigure-4.6.6に示します．

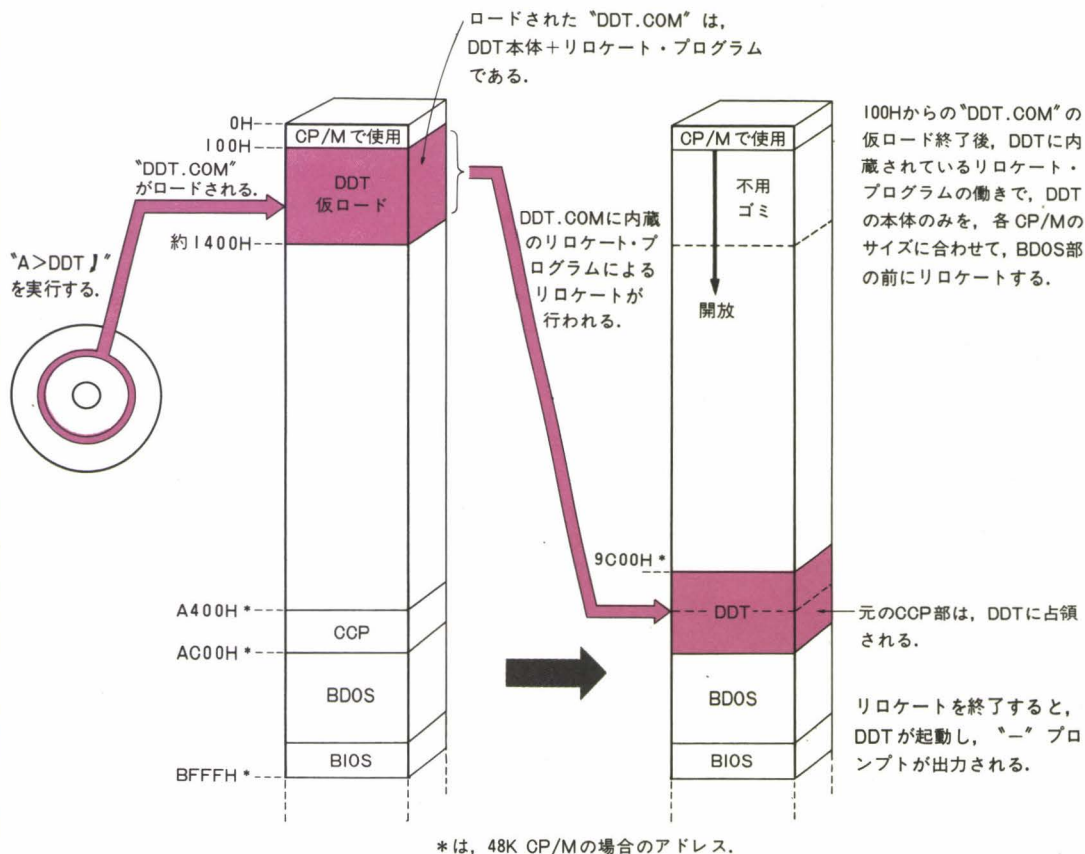



Figure-4.6.6　DDT起動の流れ

DDT の起動に関して，もう１つ知っておいた方が良いことがあります（DDT を運用する上では，全然関知する必要はありませんが）．DDT を起動することにより，アドレス0005～７Hにある，BDOSへのジャンプ・ベクトルのジャンプ先アドレスが，新しく書き替えられることと，さらにアドレス0038～AHに，Gコマンドで実行される被テスト・プログラムにブレークをかける "RST 7" のベクトルが書き込まれることです．

この DDT の起動前・起動後のメモリ・ダンプリストを，２章の Figure-2.5.2～3 に示してありますので参照して下さい．

BDOS のジャンプ・ベクトルは，DDT プログラムが位置する先頭のアドレスに書き替えられていることが分かります．これは，DDT を含む〝新CP／Mシステム〟の最下位アドレスを，外部に知らせる意味を持っています．

DDT は，8080用のデバッガです．もし，Z80をデバッグしたい場合には，デジタルリサーチ社から〝ZSID〟と言う DDT とコマンド・コンパチブルで，かつ強力な拡張機能を持った，Z80用のシンボリック・インストラクション・デバッガが別に用意されています．参考までに，ZSID によるファイル，〝SORT．HEX〟の逆アセンブル例を示しておきます．

```
A>ZSID SORT.HEX ↲

ZSID VERS 1.4
NEXT  PC   END
016E 0100 B9FF
#L100,121 ↲
  0100  LD   DE,0137
  0103  LD   C,09
  0105  CALL 0005
  0108  LD   DE,4000
  010B  LD   HL,4000
  010E  LD   C,00
  0110  LD   A,(HL)
  0111  CP   C
  0112  JP   Z,0125
  0115  INC  HL
  0116  LD   A,H
  0117  CP   50
  0119  JP   NZ,0110
  011C  INC  C
  011D  JP   Z,012E
  0120  LD   H,D
  0121  LD   L,E
  0122
#
```

Figure-4.6.7

Z80用デバッガの〝ZSID〟による逆アセンブル例．ザイログ・ニーモニックで表示されています．Figure-4.6.3（157ページ）と比較して下さい．

4.7　DUMP　（ディスク・ファイルの16進ダンプ・プログラム）

——DUMP program——

コマンド形式・機能

DUMP␣x : filename.ext

ドライブx：上のファイル〝filename.ext〞の内容を，HEX形式でダンプする．

注）x：は，それがログイン・ディスクの場合，省略できる．

DUMPコマンドは，CP／Mのシステム・コールなどを応用してのプログラム作成の見本である〝DUMP．ASM〞（DUMPプログラムのアセンブリ・ソース・ファイル）の付け足しのような存在であり，本命はこのソース・ファイルにあります．ソース・ファイルからは，コンソールとの入出力やファイルのアクセスの方法などについて多くのことを学ぶことができ，CP／Mによるマシン語開発者の絶好の参考となります．

DUMPコマンド（DUMP．COM）は，ソース・ファイルをアセンブルし，LOADコマンドを実行することによっても，新たに生成することができます．DUMPコマンド自身はアスキー表示部もなく，ごく初歩的なファイル・ダンプ機能しかありません．

実習　DUMP␣x:filename.ext

DUMPコマンドで自分自身の生みの親である〝DUMP．ASM〞をダンプしてみましょう．〝DUMP．ASM〞は，ドライブB上にあるとします．

```
A>DUMP B:DUMP.ASM ↵

0000 3B 09 46 49 4C 45 20 44 55 4D 50 20 50 52 4F 47
0010 52 41 4D 2C 20 52 45 41 44 53 20 41 4E 20 49 4E
0020 50 55 54 20 46 49 4C 45 20 41 4E 44 20 50 52 49
0030 4E 54 53 20 49 4E 20 48 45 58 0D 0A 3B 0D 0A 3B
0040 09 43 4F 50 59 52 49 47 48 54 20 28 43 29 20 31
     .
     .
```

Figure-4.7.1　ドライブB上のファイル〝DUMP．ASM〞をダンプする．ダンプ途中で何らかのキー入力を行うことにより，ブレークがかかりCP／Mにもどる．

4.8　SUBMIT　（バッチ処理プログラム）

――SUBMIT program――

コマンド形式・機能

1　**SUBMIT␣subfile**

　サブミット・ファイル〝subfile.SUB〟の内容をバッチ処理する．＄1，＄2…などの代用パラメータを使っていないサブミット・ファイルの場合のコマンド形式．

2　**SUBMIT␣subfile␣p1␣p2␣p3…**

　サブミット・ファイル中に代用パラメータを使っている場合のコマンド形式．実パラメータ，p1，p2…をサブミット・ファイル中の代用パラメータ，＄1，＄2,…に代入して，バッチ処理が行われる．

注）　サブミット・ファイル〝subfile.SUB〟は，必ずドライブA：にならなければならない．

実習1　SUBMIT␣subfile

実習2のサブミット・ファイル〝TEST．SUB〟の中の，＄1，＄2，……の代用記号の代りに，実名を入れればこの実習ができます．ここでは省略しますが，各自で行ってみて下さい．

実習2　SUBMIT␣subfile␣p1␣p2␣p3…

SUBMIT コマンドは，何度も同じ手順をくり返して実行しなければならない場合など，その手順を書いて，ファイル（サブミット・ファイル）しておき，実行時にそのサブミット・ファイル名をキーインするだけで，ファイルされたいくつもの手順を自動的に最後まで実行してしまうことができる，非常に便利なコマンドです．ソフトウェアの開発過程や，日常業務の決った手順（ルーチン）などを行う場合に，有効に利用されています．

また，実行に際しての自由度を増すために，サブミット・ファイルには，各種コマンドやファイルの実名を用いなくてもよく，＄1，＄2，＄3，……などのパラメータ代用記号で代用しておくことができ，実行時に，任意の実名を指定して実行させることが可能です．

実例として「ドライブA上のDUMPプログラムのソース・ファイル〝DUMP．ASM〟をエディットし，そのORGアドレス（オリジナルは100H）を任意のアドレスに変更し，それをアセンブルした後DDTでメモリにロードして，そのロード状態を確認する」一連の手順をサブミットしてみましょう．

そのサブミット・ファイル（TEST．SUB）をTYPEコマンドで示します．実習の際は，この内容をEDを使ってファイルします．ファイルのエクステンションは必ず〝SUB〟でなければなりません．

Figure-4.8.1　サブミット・ファイルの内容をタイプアウトで示す．

サブミット・ファイルの1行目の〝XSUB〟は，SUBMITコマンドにコンソール・バッファの読み込み機能を与える拡張コマンドであり，これにより，EDやDDTなどの内部コマンドまでサブミット可能になります(Version 1.4にはこの機能なし)．例題としたこのサブミット・ファイルには，$1，$2，$3…の代用パラメータが実名の代りに使われていることに注目して下さい．

さて，このサブミット・ファイル（TEST．SUB）を実行してみましょう．オリジナルのDUMP．ASMのORGは100Hです．これを2500Hに変更するように代用パラメータの指定をして実行します．

$1＝100
$2＝^Z（Ctrl-Zをキーインする）
$3＝2500

と，コマンドラインに記述します．

```
A>SUBMIT TEST 100 ^Z 2500 ┘ -----この1行のキーインで以下のすべてが自動的に実行される.
                                 "^Z" は "^" + "Z" ではなく, Ctrl-Zそのものをキーインすること.
                     ↑    ↑    ↑
A>XSUB               $1   $2   $3

A>ED DUMP.ASM
     : *0A
    1: *10T
    1:  ;           FILE DUMP PROGRAM, READS AN INPUT FILE AND PRINTS IN HEX
    2:  ;
    3:  ;           COPYRIGHT (C) 1975, 1976, 1977, 1978
    4:  ;           DIGITAL RESEARCH
    5:  ;           BOX 579, PACIFIC GROVE
    6:  ;           CALIFORNIA, 93950
    7:  ;
    8:              ORG      100H
    9:  BDOS        EQU      0005H    ;DOS ENTRY POINT
   10:  CONS        EQU      1        ;READ CONSOLE
    1: *FORG        100H    ------文字列サーチ. EDの内部コマンドも実行可能であることに注目.
    8: *0LT
    8:              ORG      100H
    8: *S100 2500          ---------文字列の置き替え.
    8: *0TT
    8:              ORG      2500H -------100が2500に変更された. 新ORG.
    8: *E ---- EDを終る.

(xsub active)
A>ASM DUMP   -----アセンブル.
CP/M ASSEMBLER - VER 2.0
2657
002H USE FACTOR
END OF ASSEMBLY  -----アセンブル終了.

(xsub active)
A>DDT    ---------DDTを起動.
DDT VERS 2.2
-IDUMP.HEX ┐
           │ HEXファイルのロード.
-R         │ DDTの内部コマンドも実行可能であることに注目.
NEXT  PC   ┘
2613 0000

-D2500   ------新ORGアドレスのダンプ.
2500 21 00 00 39 22 15 26 31 57 26 CD C1 25 FE FF C2 !..9".&1W&..%...
2510 1B 25 11 F3 25 CD 9C 25 C3 51 25 3E 80 32 13 26 .%..%..%.Q%>.2.&
2520 21 00 00 E5 CD A2 25 E1 DA 51 25 47 7D E6 0F C2 !.....%..Q%G}:..
   .
   .
2590 0F 0F 0F 0F CD 7D 25 F1 CD 7D 25 C9 0E 09 CD 05 .....}%..}%.....
25A0 00 C9 3A 13 26 FE 80 C2 B3 25 CD CE 25 B7 CA B3 ..:.&....%..%...
25B0 25 37 C9 5F 16 00 3C 32 13 26 21 80 00 19 7E B7 %7._..<2.&!...~.

-L2500   ------逆アセンブル.
  2500  LXI  H,0000
  2503  DAD  SP
  2504  SHLD 2615
```

```
      .
      .
  2515   CALL  259C
  2518   JMP   2551
  251B   MVI   A,80
-G0         DDTを終る.

(xsub active)
A> ----------SUBMIT実行全部終了.
```

Figure-4.8.2　サブミット・ファイルの実行.
ファイルの内容がすべて自動的に実行されて行く.

このように，サブミット・ファイル内の一連の処理が，連続して実行されました．今回は ORG 100 H を ORG 2500H に変更しましたが，＄1と＄3の値を変えることにより，任意の ORG を指定して一連の処理を行わせることができるわけです．

＄1，＄2，…などの代用パラメータは，ファイル名，各種コマンド，数値，データなどの代用として使いますが，実行に際して変化することなく一定のものは，実際の名称なり，数値なりを書いておく方が実行時に楽です．

Figure-4.8.1 のサブミット・ファイルの2行目には，実ファイル名が書かれていますが，もしこれを代用パラメータにしておき，実行時に任意のファイルを指定したい場合，このサブミット・ファイル（TESTX．SUB）は次のようになります．

```
A>TYPE TESTX.SUB
XSUB
ED $1           実ファイル名を使わずに，＄1とした.
0A
10T
FORG     $2H
0LT
S$2$3$4
0TT
E
ASM DUMP        Figure-4.8.1と同様の処理.
DDT
IDUMP.HEX
R
D$4
L$4
G0

A>
```

Figure-4.8.3　ファイル名を代用パラメータに変更した例

この場合の実行時のコマンド・ラインは，例えば次のようになります．

となります．これは自動的にソース・ファイル〝DUMP．ASM〟の ORG xxxxH を ORG yyyyH に変更して，一連の処理を行わせるコマンドになります．

4.9 SYSGEN (CP/Mシステム生成プログラム)

——SYStem GENerator——

コマンド形式・機能

SYSGEN

ディスケットのシステム・トラックの内容をメモリにロードしたり，メモリ上のCP/Mシステムをディスケットのシステム・トラックに組み込んだりする．日常は，システム・ディスクのコピーに使われる．SYSGENが起動した後は，SYSGENが出力するメッセージに従う．

注） 8インチの標準ディスク以外は，それぞれのマシン専用のプログラムであり，他のコマンドのように汎用性はない．

実習 システム・ディスクのコピー

```
A>SYSGEN ↲

SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)A
SOURCE ON A, THEN TYPE RETURN↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B
DESTINATION ON B, THEN TYPE RETURN↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)↲

A>
```

Figure-4.9.1 ドライブA上のディスクのCP/Mシステムを，ドライブB：上のディスクへコピーする例．

SYSGENの起動後，ソース側あるいはデスティネーション側，どちらのドライブでも自由にディスケットを入れ替えることができます．そして，SYSGENプログラムから出力されるメッセージに従って，どちらのドライブからどちらのドライブへでも自由にCP/Mシステムを何回でもコピーすることが可能です．

SYSGEN を使った BIOS の変更や新システムの組み込みについては，次の「MOVCPM」で実習します．

解説 SYSGEN

SYSGEN コマンドは，対象が〝ファイル〟ではなく，システム・トラック上の〝CP／M システム〟であるために，ディスケット上の物理的なトラック#，セクタ#と密接な関係があります．よって，8インチ標準ディスケットの SYSGEN コマンド（SYSGEN．COM）は，8インチ標準ディスケットの専用であり，両面倍密度や，ミニディスクなどでは使用できません．

同様に，各社のパーソナル・コンピュータの CP／M の SYSGEN コマンドは，その機種の専用に作られたものであり，他の機種では使用できません．この点が，マシンの種類に関係なく，完全に互換性のある通常のコマンド（プログラム）と異なりますので注意して下さい．

SYSGEN コマンドは，デジタルリサーチ社のマニュアルには書かれていませんが，次のような使い方ができるようです．CP／M システムのイメージ・ファイル（BIOS と BOOT を含んだもので，この例では〝32KCPM．COM〟）があらかじめファイルしてあれば，次の手順で簡単にそのシステム・ディスクを作ることができます．

```
A>SYSGEN 32KCPM.COM ↵  -------SYSGENに続けて，ファイル化したCP/Mシステムのファイル名をキーインする.
                              (ログイン・ディスクでない場合は，ドライブ名を付ける).
SYSGEN VER 2.0
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B -----通常はソース・ドライブ名の問いが出力され
DESTINATION ON B, THEN TYPE RETURN↵                るが，省略されていることに注目.
FUNCTION COMPLETE  -------ドライブB：のディスケットに，CP/Mシステムが組み込まれた.
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)↵

A>
```

Figure-4.9.2　マニュアルには載っていない SYSGEN コマンドの使い方．システム・トラックからコピーするのではない点に注目．

ここで使用する，BIOS と BOOT を含んだ CP／M システムのメイージ・ファイルは，例えば，Figure-4.10.5の最後の SYSGEN コマンドの実行を行う直前，あるいは，行った直後に〝SAVE 34 32KCPM．COM〟を行えば作成することができます（セーブするファイル名は任意）．

また，すでにディスク上のシステム・トラックに組み込んであるものは，SYSGEN を起動し，ソース・ディスクとして，そのシステムを読み出し，そのまま〝リターン〟して CP／M にもどした後，上記の SAVE コマンドを実行すれば，〝ファイル〟として CP／M システムを持つことができます．

この方法を Figure-4.9.3に示します.

```
A>SYSGEN↲ -----SYSGENを起動.
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)B ----ドライブB：上のシステムをファイル化したい場合.
SOURCE ON B, THEN TYPE RETURN↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)↲ ----リターンをキーインして，そのまま
                                                                          CP/Mに戻る.
A>SAVE 34 B:32KCPM.COM↲ ---SAVEコマンドの実行.“34”は8インチ標準ディスクの場合のページ数.

A>STAT B:32KCPM.COM↲ ------SAVEされたファイルの確認.

 Recs  Bytes  Ext Acc
   68     9k    1 R/W B:32KCPM.COM -----システム・トラックから，CP/Mシステムが，
Bytes Remaining On B: 58k              ファイル“32K CPM.COM”としてセーブされている.

A>
```

Figure-4.9.3 システム・ディスクから CP/M システムを〝ファイル〞としてセーブする例.

但し，以上は8インチの標準ディスクについてのみ可能なことであり，他の場合は，SYSGEN プログラムの作り方によっては，できないものもありますのでご注意下さい.

4.10　MOVCPM　(CP/Mシステム・リロケート・プログラム)

——MOVe CP/M system——

コマンド形式・機能

1　**MOVCPM␣ss␣***

メモリサイズssKバイトのCP／Mシステム(インテルMDS用以外のBIOSとBOOTは含まれない）を生成し，SAVEコマンドの実行が可能なように，メモリ上に配置する．ssは20～64の範囲（単位はKバイト）でなければならない．

2　**MOVCPM␣*␣***

コンピュータのRAMのリード／ライト・テストを行ってRAMエリアの最大アドレスを求め，設定可能な最大のメモリサイズのCP／Mシステムを生成する．その他は同上の機能を持つ．

3　その他のコマンド形式は，一般のCP／Mマシンでは，まったく実用不可能であるので省略．

注）　2の形式の場合，最大のRAMエリアを自動的に求めてしまうので，その範囲にVRAMやワークエリアなどが含まれている場合，このコマンド形式は，使うことができない．注意を要する．

メモリサイズに関してはFigure-2.1.2を参照．

MOVCPMコマンドはそれだけを実行しても，一般のCP／Mマシンでは何の意味もないので，ここでは新しいCP／Mシステムを作成する実習を行い，その過程でMOVCPMを理解できるように解説します．

解説　MOVCPMによる新CP／Mシステムの作成

MOVCPMコマンドは，それ自身でCP／Mシステム（BIOS部とBOOT部を除く）をリロケータブルなオブジェクトの形式で持っており，それをユーザーが指定する任意のCP／Mサイズに合致するよう，自動的にリロケートを行うコマンドです．コマンド形式1，2の場合，とりあえずメモリに置かれるアドレスは実アドレスではなく，SYSGENコマンドにより，そのままディスクのシステム・トラックに組み込み可能な位置（TPAの下位）に置かれます．

しかし，TPAにロードされた新サイズのメモリ・イメージは，まだBIOSとBOOTのモジュー

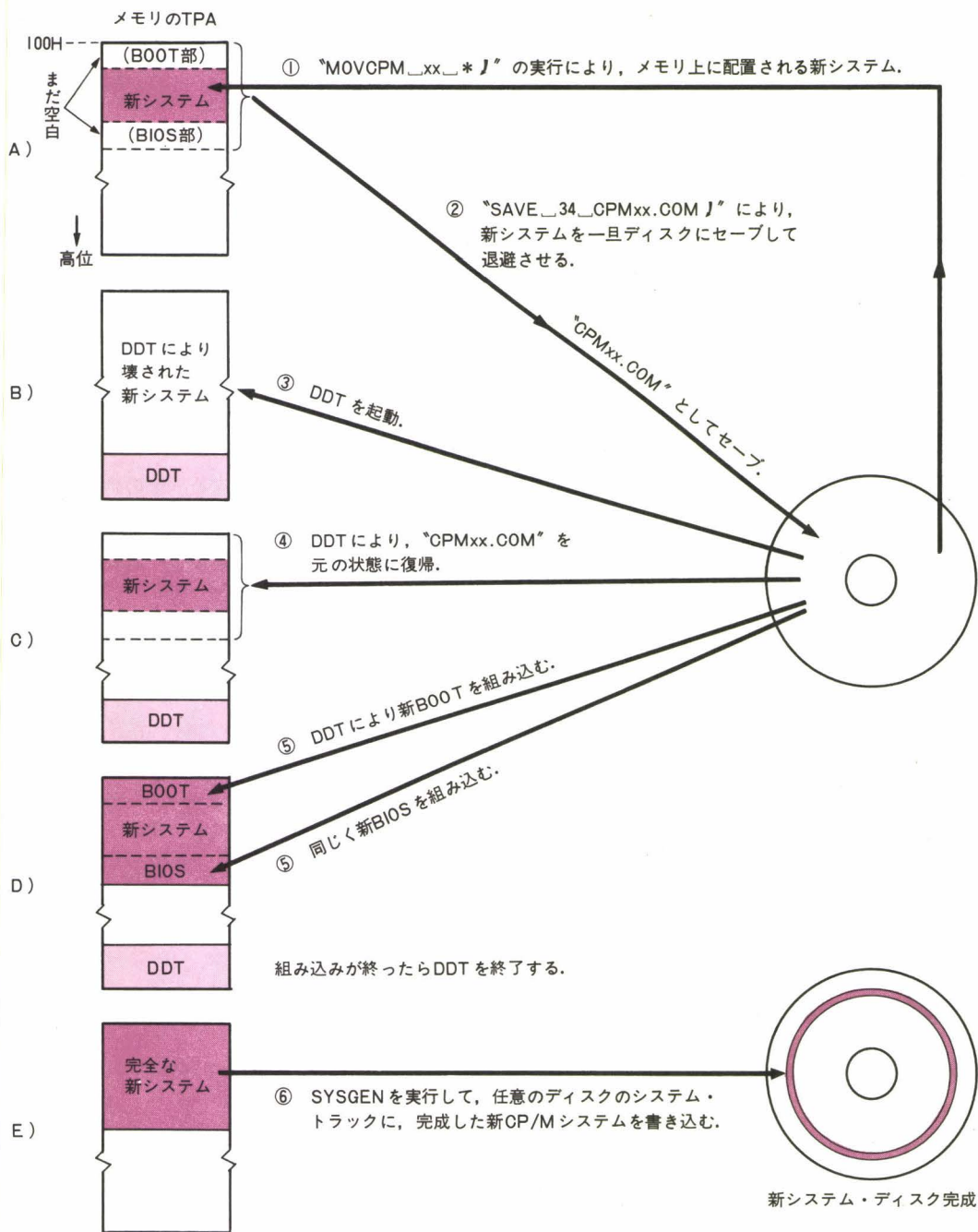



Figure-4.10.1 新CP／Mシステムの作成手順

ルが組み込まれていません．これらを組み込むためにはDDTを使いますが，DDTが起動する際に，せっかくロードされた新システムのイメージを壊してしまいます(「DDTコマンド」参照)．これを救うためにシステムを一旦SAVEして，ディスク上に退避しておく必要があります．MOVCPMが実行された時に表示されるメッセージ（後で示す）に，〝SAVE 34 CPMxx．COM〟とあるのはこのためなのです．

新システム・イメージをSAVEした後，DDTを起動して新システムを再ロードし，その上から新BIOSと新BOOTを組み込みます．組み込みが終ればDDTを終了して，出来上った完全なCP／Mシステムを SYSGEN でディスクのシステム・トラックに組み込み，新システム作成の作業は完了となるわけです．

以上の図解を Fig-4.10.1 に示します．この図では，新BOOTと新BIOSのHEXオブジェクト・ファイルは，あらかじめアセンブルにより，すでにディスク上に存在しているものとして話を進めていますが，実習ではこのアセンブルをDDTを起動する直前に行っています．

実習1 MOVCPM␣ss␣*

現在のCP／Mサイズが48Kであるとして，新たに32Kサイズに縮小したCP／Mシステムを作ってみましょう．手順は少々複雑でMOVCPM, SAVE, ED, REN, ASM, DDT, SYSGENの各コマンドを駆使しなければなりません．しかしまず前提として，それぞれのCP／Mマシン用のBIOSとBOOTのアセンブリ・ソース・ファイルが付属していなければ，CP／Mのメモリサイズを変更することは不可能です．

```
A>MOVCPM 32 * ↲

CONSTRUCTING 32k CP/M vers 2.2
READY FOR "SYSGEN" OR
"SAVE 34 CPM32.COM" ------このメッセージ通りに次は，SAVEコマンドを実行する.

A>SAVE 34 CPM32.COM ↲ ----- 生成されたRAM上の32K CP/Mシステムを一旦セーブする.
                            "34" はCP/Mシステムの容量(ファイルする場合の)が(34×256)=8.5Kバイトであるた
A>STAT CPM32.COM ↲ -----------セーブされたファイルの確認.      めで，サイズに関係なく常に34である.
 Recs  Bytes  Ext Acc
   68     9k    1 R/W A:CPM32.COM -----このファイルが32K CP/Mの，BIOSとBOOTを
Bytes Remaining On A: 88k                                     除いたイメージである.

A>
```

Figure-4.10.2 32K CP／Mシステムの生成と，そのイメージに対するSAVEコマンドの実行，さらにセーブされたファイルのSTATによる確認．

以上の手順で，BIOS と BOOT のモジュールを除いた 32K CP／M システムのイメージが，〝CPM 32．COM〞としてディスク上にセーブされました．Fig-4.10.1 の図も随時参照して下さい．以上の操作は図解のA）に当ります．

もし MOVCPM の実行で，次のようなエラー・メッセージが出力されて，コンピュータが HLT 状態（実行停止状態）になった場合は，現在働いている CP／M のシステムと実行した MOVCPM のプログラムが，同一ディスクからのものではなかったからです．これは，CP／M システムと MOVCPM プログラムの双方にキーワードがあり，その両者が一致しないと MOVCPM は実行できないことを示しています．正規に購入した CP／Mでは，もちろん両者のキーワードは一致しています．

```
A>MOVCPM 32 *↲

CONSTRUCTING 32k CP/M vers 2.2
SYNCRONIZATION ERROR
```

Figure-4.10.3
シンクロ・エラーが発生した例.
HLT 状態になっているので，回復にはリセット・ボタンによるか，割り込みによるかいずれかである．

次に，BIOS と BOOT のソース・プログラムの CP／M サイズを変更して，再アセンブルし，32K CP／M用のBIOS と BOOT の HEX オブジェクト・ファイルを作成します．通常 BIOS や BOOT のソース・プログラムは，CP／M サイズがEQU 命令で指定されており，この部分を任意のサイズ(20～64)に変更・再アセンブルすることにより，容易に任意のサイズ用のBIOS や BOOT の HEX ファイルを作り出すことができます．この例で用いる BIOS と BOOT のソース・プログラムには，〝MSIZE EQU 48〞と書かれてあり，エディタでこの48を32に変更してアセンブルすればよいわけです．

- ED でソース・ファイルの48を32に変更．
- REN コマンドでファイル名の48を32に変更．
- アセンブラの実行．

を，BIOS，BOOT の双方について行った実行例を次に示します．

```
A>ED PTBIOS48.ASM↲ --------EDを起動.
     : *NMSIZE^ZOTT↲ ----文字列"MSIZE"をサーチ．その行をタイプアウト.
   30:  MSIZE   EQU  48            ;MEMORY SIZE IN KBYTES.
   30:  *S48^Z32^ZOTT↲----48を32に置き替えて，その行をタイプアウト.
   30:  MSIZE   EQU  32            ;MEMORY SIZE IN KBYTES.
   30:  *E↲ ------EDを終了する.

A>REN PTBIOS32.ASM=PTBIOS48.ASM↲ ---- 48を32とリネーム（しておいた方が今後のためによい).
```

```
A>ASM PTBIOS32 ↲ -------アセンブラの実行.
CP/M ASSEMBLER - VER 2.0
7EC1
00BH USE FACTOR
END OF ASSEMBLY -------アセンブラの終了.

A>ED PTBOOT48.ASM ↲ ------BIOSと全く同様なことを，BOOTについて行う.
     : *NMSIZE^ZOTT ↲
   18:  MSIZE    EQU  48         ;MEMORY SIZE IN DECIMAL KB.     **
   18: *S48^Z32^ZOTT ↲
   18:  MSIZE    EQU  32         ;MEMORY SIZE IN DECIMAL KB.     **
   18: *E ↲

A>REN PTBOOT32.ASM=PTBOOT48.ASM ↲

A>ASM PTBOOT32 ↲
CP/M ASSEMBLER - VER 2.0
0080
002H USE FACTOR
END OF ASSEMBLY

A>      以上で32K CP/M用のBIOSとBOOTのHEXオブジェクトが出来上った.
```

Figure-4.10.4　32K CP／M サイズ用の BIOS と BOOT の HEX ファイルを作る作業.

以上の作業で，新 BOOT と BIOS の HEX オブジェクト・ファイルが出来たので，次は，先程 SAVE した新32Kシステムを再びメモリ上にロードして，BOOT と BIOS を組み込む作業を DDT を使って行います．この操作は図解のB)，C)，D) に当ります．

```
A>DIR PT???32.HEX ↲ ------ BOOTとBIOSのHEXファイルの確認.
A: PTBIOS32 HEX : PTBOOT32 HEX

A>DDT CPM32.COM ↲ ----------- ディスク上に退避させておいた新システムのイメージを，メモリ上に戻す.
DDT VERS 2.2
NEXT  PC
2300 0100

-IPTBIOS32.HEX ↲  } 32K CP/M用のBIOSのHEXオブジェクト（ORGは7A00H）を，
-RA580 ↲          } RコマンドにA580Hのバイアス・アドレスを付けてメモリにロードする.
NEXT  PC            結局，32K用のBIOSはアドレス1F80Hからロードされる.
2300 0000

-IPTBOOT32.HEX ↲  } BOOTのORGは0であり，またCP/Mサイズによらず，
-R900 ↲           } ロード・アドレスは900Hに一定している.
NEXT  PC            Rコマンドに900Hのバイアスを付けてロードする.
2300 0000

-H7A00,A580 ↲ ------ HEXの計算．ORG=7A00HのHEXファイルを，A580Hを加算してロードした場合，
1F80 D480            FFFFHを過ぎて，1F80Hからロードされることになる.
```

```
-D1F70,1F9F ↲ ------ロードされたBIOSの先頭部の確認.
1F70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
1F80 C3 9C 7A C3 F8 7A C3 51 7B C3 5A 7B C3 8D 7B C3 ..z..z.Q{.Z{..{.
1F90 59 7D C3 64 7D C3 69 7D C3 F1 7B C3 B2 7B C3 F3 Y}.d}.i}..{..{..
-L1F80 ↲ -----------逆アセンブルでは, BIOS先頭のジャンプ・ベクトルが見られる.
  1F80  JMP  7A9C
  1F83  JMP  7AF8
  1F86  JMP  7B51
  1F89  JMP  7B5A
     .
     .
  1F9E  JMP  7BF3

-D900,92F ↲ ------- 同じくBOOTの確認.
0900 1E 0A 31 00 01 21 00 64 16 33 0E 02 06 04 79 CD ..1..!.d.3....y.
0910 2A 00 15 CA 00 7A 06 00 0C 79 FE 1B DA 0F 00 3E *....z...y.....>
0920 53 D3 E8 DB EC 0E 01 C3 0C 00 D3 EA CD 41 00 3E S............A.>
-L900 ↲ -------逆アセンブル.
  0900  MVI  E,0A
  0902  LXI  SP,0100
  0905  LXI  H,6400
  0908  MVI  D,33
     .
     .
  0916  MVI  B,00
-^C -------DDTを終了する. G0 ↲ でもよい.

A>SYSGEN ↲ -------メモリ上に完成している新CP/Mシステムを, ディスクのシステム・トラックに書き込む.
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)↲ ----ここは必ずリターン・キーでスキップすること.
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B ----B: 上のディスクに書き込む場合.
DESTINATION ON B, THEN TYPE RETURN↲ ---書き込みの実行.       任意のドライブ名を指定できる.
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)↲ ----ドライブ名を指定すれば,
                                                   さらに何枚でもコピー可能である.
A>
```

Figure-4.10.5 DDT による新システムへの BIOS と BOOT の組み込みと, メモリ上に完成した新 CP／M システムのSYSGEN によるディスク上のシステム・トラックへの書き込み.

このあとに, "SAVE 34 32KCPM.COM" を実行して, 32K CP／M の完全なイメージ・ファイルを作っておくと便利でしょう (「解説 SYSGEN」を参照).

以上一連の作業で, 32K CP／Mのシステム・ディスケットが出来上りました. これをドライブAに入れて, 起動してみましょう.

```
リセット
32K CP/M ver2.2
A>B:DDT ↵
DDT VERS 2.2
-D0,F ↵
0000 C3 03 7A 00 00 C3 00 5C 16 33 0E 02 06 04 79 CD ..z....¥.3....y.
-^C
A>
```

Figure-4.10.6　完成した32K CP／Mのシステム・ディスケットによる起動.ついでにアドレス00～0FHをDDTでダンプして，00Hと05Hのジャンプ・ベクトルのジャンプ先アドレスに注目.次に示す48Kの場合と比較.

参考までに従来の 48K CP／M の場合を示しておきます.

```
リセット
48K CP/M ver2.2
A>DDT ↵
DDT VERS 2.2
-D0,F ↵
0000 C3 03 BA 00 00 C3 00 9C 16 33 0E 02 06 04 79 CD .........3....y.
-^C
A>
```

Figure-4.10.7　従来の 48K CP／M の場合.
00Hと05Hのジャンプ先のアドレスに注目，上記32Kの場合と比較.

ここでもう1つの参考として,32K用 BIOS のアセンブルにより生成された,PRN ファイルの ORG 命令の付近のリストを示します.

```
A>TYPE PTBIOS32.PRN ↵
          .
          .
          .
7A00                ORG   BIOS              ;START OF CBIOS STRUCTURE.
 注目.           ;
                 ; I/O JUMP VECTOR
                 ; THIS IS WHERE CPM CALLS WHENEVER IT NEEDS
                 ; TO DO ANY INPUT/OUTPUT OPERATION.
                 ; USER PROGRAMS MAY USE THESE ENTRY POINTS
                 ; ALSO, BUT NOTE THAT THE LOCATION OF THIS
                 ; VECTOR CHANGES WITH THE MEMORY SIZE.
                 ;
7A00 C39C7A         JMP   BOOT              ;FROM COLD START LOADER.
7A03 C3F87A  WBOOTE: JMP  WBOOT             ;FROM WARM BOOT.
7A06 C3517B         JMP   CONST             ;CHECK CONSOLE KB STATUS.
7A09 C35A7B         JMP   CONIN             ;READ CONSOLE CHARACTER.
```

```
7A0C C38D7B          JMP   CONOT              ;WRITE CONSOLE CHARACTER.
7A0F C3597D          JMP   LIST               ;WRITE LISTING CHAR.
7A12 C3647D          JMP   PUNCH              ;WRITE PUNCH CHAR.
7A15 C3697D          JMP   READER             ;READ READER CHAR.
7A18 C3F17B          JMP   HOME               ;MOVE DISK TO TRACK ZERO.
7A1B C3B27B          JMP   SELDSK             ;SELECT DISK DRIVE.
7A1E C3F37B          JMP   SETTRK             ;SEEK TO TRACK IN REG A.
7A21 C3497C          JMP   SETSEC             ;SET SECTOR NUMBER.
7A24 C3547C          JMP   SETDMA             ;SET DISK STARTING ADR.
          .
          .
          .
          .
```

Figure-4.10.8　32K用BIOS の PRN ファイルのリストの一部．ORG 命令の付近．

Figure-4.10.6～8 のリストから CP／M のメモリ・サイズによって，各モジュールのリロケート先が異なることが理解されたと思います．48Kと32Kでは16Kバイトの差があり，当然のことながら32K CP／M ではその BIOS も16Kバイト分下位にリロケートされています．

ここで，各種メモリ・サイズの BIOS をシステムに組み込む際（図解のDの作業時）のバイアス値（Figure-4.10.5 では A580H であった）の一覧表を示しておきます．

CP/Mサイズ	バイアス値(HEX)	CP/Mサイズ	バイアス値(HEX)	CP/Mサイズ	バイアス値(HEX)
20 K	D 580	36 K	9580	52 K	5580
24 K	C 580	40 K	8580	56 K	4580
28 K	B 580	44 K	7580	60 K	3580
32 K	A 580	48 K	6580	64 K	2580

Figure-4.10.9　BIOS組み込み時の各メモリ・サイズに対するバイアス値

実習2　BIOSの変更・システムの組み込み

次に，CP／M のメモリ・サイズは変更せず，現在の CP／M の BIOS のみを変更して，システムに組み込む場合の手順を示しておきます．BIOS に新しい機能を追加して，システム・ディスケットを作りたい場合，以下の手順で作成することができます．メモリ・サイズの変更よりは，こちらの方がよく行われる作業でしょう．

```
A>ED NOWBIOS.ASM ↲ -----現在のBIOSのアセンブリ・ソース・ファイルに対しEDを起動する.
*          .
           .        } BIOSの変更作業を行う.
*          .
*E ↲

A>REN NEWBIOS.ASM=NOWBIOS.ASM ↲ ---新BIOSのソース・ファイルをリネームしておく方がよい.

A>ASM NEWBIOS ↲ ----新BIOSのアセンブル実行.
    .
    .

A>SYSGEN ↲ ------現在のCP/Mのメモリ・イメージを得るために, SYSGENを実行.
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)A ----ドライブA:に現在のシステム・ディスクがある.
SOURCE ON A, THEN TYPE RETURN ↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) ↲ --- そのままの状態でCP/Mに戻る.

A>SAVE 34 NOWCPM.COM ↲ ------メモリ上の現在のCP/Mのイメージをディスクにセーブする.
                                    「34」は標準CP/Mのシステムの大きさ.
A>DDT NOWCPM.COM ↲ ------セーブしたCP/Mイメージを, 再度DDTでメモリ上にロードする.
DDT VERS 2.2
NEXT  PC
2300 0100
-INEWBIOS.HEX ↲  } 新BIOSのHEXオブジェクトを組み込む.
-Rxxxx ↲         } xxxxのバイアス値はCP/Mサイズによって異なる. 表を参照.
-^C  ------ DDTを終わる.

A>SYSGEN ↲ ------ 再びSYSGENを起動.
SYSGEN VER 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP) ↲ ------ここは必ずリターンでスキップすること!
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)B -----システムを書き込むディスクを指定.
DESTINATION ON B, THEN TYPE RETURN ↲
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT) ↲

A>   ドライブB:上に新CP/Mのシステム・ディスクが出来上り.
```

Figure-4.10.10 現在のCP／MのBIOSを変更して，システムに組み込む手順.

注) CP／MによってはBIOSのアセンブリ・ソース・ファイルが公開されていなかったり, SYSGENコマンドがオリジナルの機能を持っていなかったりして，CP／MサイズやBIOSの変更ができないものが少なくありません．それぞれのCP／Mのマニュアルをご覧下さい．

5章　CP/Mによるマシン語開発実習

本章では，CP／Mのみを使って，アセンブラによる8080のマシン語プログラムを作成する実習を行います．

8080のマシン語は，現在の多くのCP／Mマシンで使われているZ80上でもそのまま動作することは，よくご承知のことでしょう．

ソフトウェアを開発する場合，CP／Mの持つ各コマンドの機能を，フルに活用しなければなりません．CP／Mに含まれているビルトイン・コマンドや，ユーティリィー・プログラム（トランジェント・プログラム）のほとんどは，CP／Mでソフトウェアを開発する場合の支援ツールなのです．どのような時に，どのコマンドをどう使うか．このKnow Howはそれぞれのユーザーが，経験を積んで修得するほかはないのかも知れません．しかし本書により，各コマンドの機能の詳細は理解することができます．後はそれをどう応用するかの問題なのです．

では，プログラムのフローチャートの作成→ソース・ファイルの作成→アセンブル→デバッグ→完成までの全手順を実習してみましょう．

5.1　作成するプログラムの目的

ソート・プログラムを作ってみましょう．ソートは，事務用アプリケーション・プログラムなどで欠くことのできない機能であり，ABC順や，アイウエオ順に文字列を並べ替えたり，数値を小さい（または大きい）順に並べ替えたりする機能のことです．

今回作成するプログラムは，その最も基本的なものであり，メモリ上の1バイトの値（00～FFH：0～255）を，小さい順に並べ替える作業を行わせるものです．プログラムの仕様を次に示します．

実習で作成するプログラムの仕様

> メモリ上の，アドレス4000Hから4FFFHまでの4Kバイトにストアされている1バイト単位のデータ（00～FFH）を，小さい順に同じエリアのメモリ内で並べ替えよ．
> このプログラムが実行された時には，プログラム実行開始のメッセージを，終了の時には，プログラム終了のメッセージをコンソールに出力せよ．また，プログラム終了と同時にCP／Mに戻ること．

要するに，アドレス4000H～4FFFHの4KバイトのRAM上の無秩序なデータ（1バイト単位）を，同じエリア内で小さい順に並べ替えればよいのです．

では，このプログラム名を〝SORT〟として，作業に取り掛かりましょう．作業開始から完成までの基本的な手順を図示しておきます．

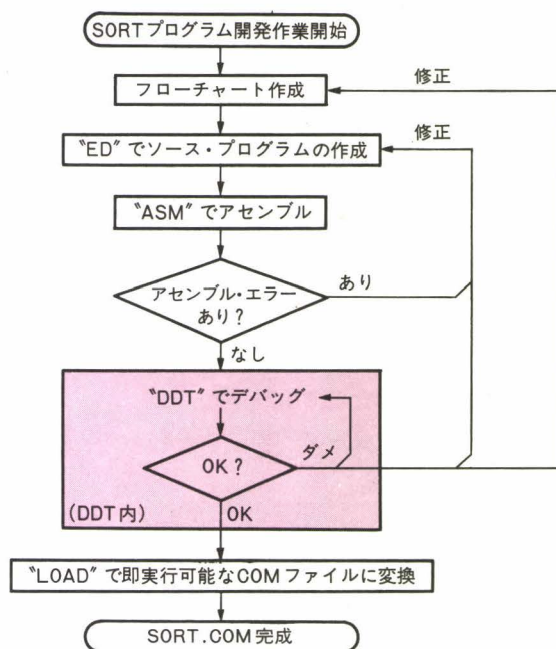



Figure-5.1.1　〝SORT″プログラム開発手順

5.2　フローチャートの作成

読者の中には，この程度のものならフローも何もいらず，キーボードに向って直接ソース・プログラムを叩き込んで行ける方も多いと思います．実は筆者も確たる見透しも立っていないのに〝フローチャートよりキーボード″の誘惑に負けてしまう一人なのです．しかし，フローチャートを書くことこそ，ソフトウェア開発を能率よく最短時間で行うための大原則なのです．フローチャートを書くことは時間がかかってめんどうな仕事ですが，結局それが最短コースであることは誰もが認めるところです．簡単なプログラムであっても，極力フローを書きましょう．

さて，この〝SORT″プログラムを本書の例題向きに，筆者が考えたフローチャートを，Figure-5.2.1に示します．このプログラムを実現する手法は，他にも幾通りもあると思います．のちほど，読者自身も別のやり方でデザインされることをお勧めします．

このプログラムのフローの要点を解説しておきましょう．

(1)　まずポインタとして，D，EおよびH，Lレジスタに，ソート・データ・エリアの先頭アドレス4000Hをセットしておきます．

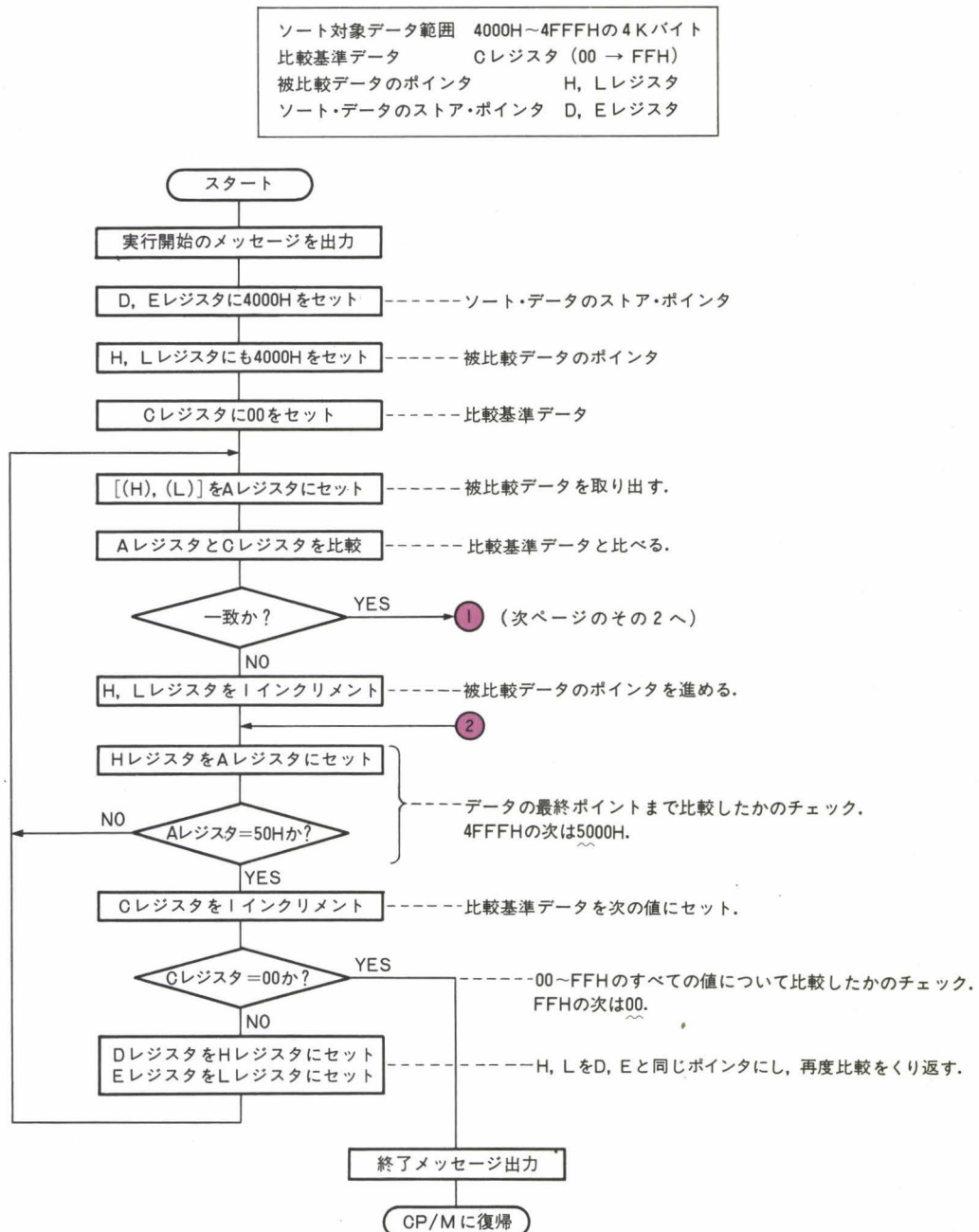



Figure-5.2.1•a　"SORT"プログラムのディーテイル・フローチャート（その1）

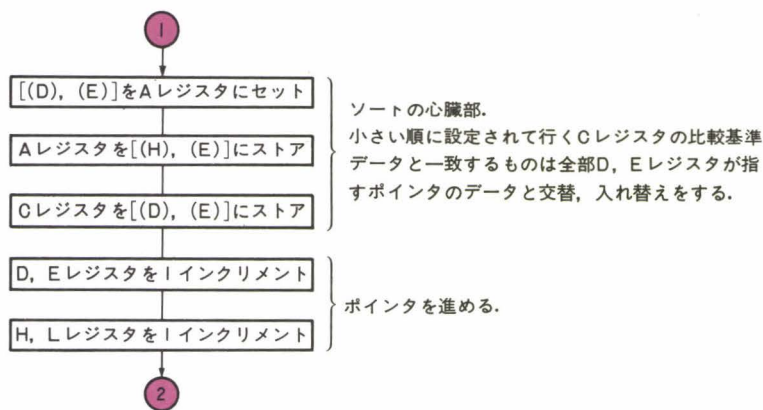



Figure-5.2.1·b 〝SORT〟プログラムのディーテイル・フローチャート(その2)

(2) Cレジスタに，ソートするデータの最小の値である00をセットします．この値はループごとに次々とインクリメントされます．

(3) 4000Hから始まるH，Lポインタを次々と進めながら，H，Lレジスタでアドレスされるメモリの内容と，Cレジスタの値を比較して行きます．

(4) もし一致したデータが発見されたら，そのデータをD，Eレジスタでアドレスされるメモリにストアします．ストアを行う前に，そこにあったデータを，一致したデータがあったアドレス(H，Lポインタ)に移し替えておきます．入れ替え処理を行った後，D，EおよびH，Lポインタをそれぞれ1つ進めておきます．

(5) 同様に，H，Lポインタを次々と進めながら，H，Lレジスタでアドレスされるメモリのデータと，Cレジスタの値を比較して行きます．一致したデータが発見されたら再び 4)の処理を行います．

(6) データ·エリアの最後（4FFFH）まで比較検査したら，次はCレジスタの値を1つインクリメントして，5)のループを，今度は1つ大きい新しいデータ（Cレジスタの値）を基にくり返します．

(7) このようなことを同様にくり返し，Cレジスタの値が，最大データであるFFHを過ぎると，全データのソートが完了したことになり，このプログラムは終了します．

以上の要点を図化したものを Figure-5.2.2に示しておきます．なお，メッセージ出力の部分は〝システム·コール〟と呼ばれる機能を使いますが，これについての詳細は，次巻「応用CP／M」で解説します．

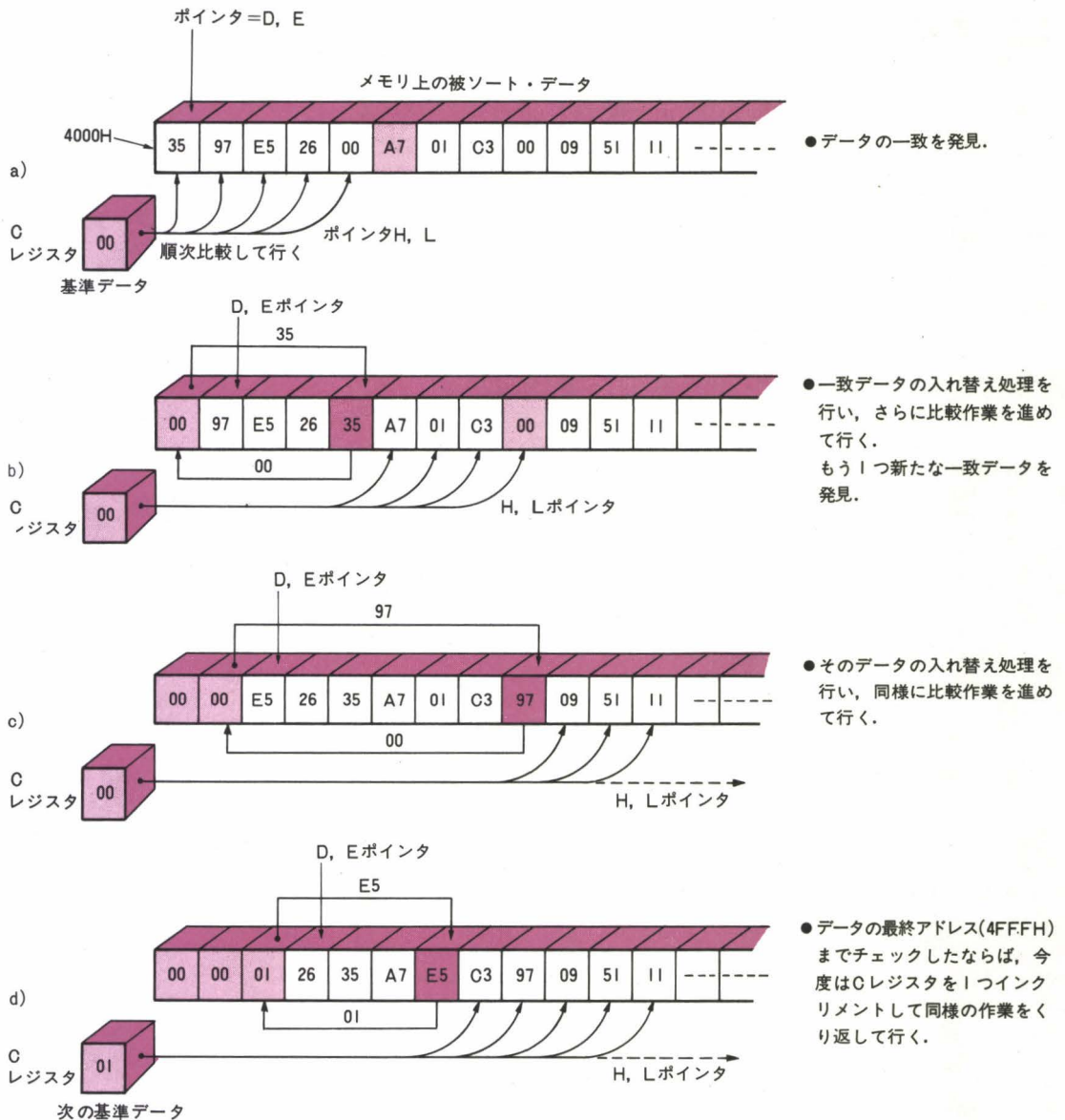



Figure-5.2.2　Cレジスタを使った実際のソート手順

5.3 アセンブリ・ソース・ファイルの作成

使用レジスタや，その目的などをあらかじめ，きちんと定めてから，ソース・ファイルの作成に取りかかります．深く考えなければならない部分は紙の上に書いて検討しますが，そうでないものはどんどんキーボードから叩き込んで行けばよいでしょう．但し，その場合，プリンタ用紙をケチらずにふんだんに使用すること！　リストの変更があった場合は，たとえ僅かな変更でも，どんどん新しいリストをプリントアウトすること．これがミスの少ないソース・ファイルを早く作成する秘訣です．

使用するディスケットは，新しいディスケット（物理的に新しいという意味ではなく，〝クリアな〟ということ）上にＣＰ／Ｍシステムと共に，少なくとも次の５つのプログラム・ファイルをコピーしておき，そのディスケット上で各作業を行うのが能率的でしょう．

```
A>DIR ↲
A: ED       COM : ASM      COM : DDT      COM : LOAD     COM
A: STAT     COM
A>
```

Figure-5.3.1　プログラム開発のために，ディスク上に用意しておくファイル．

では，ED によりソース・ファイルを作成する作業を始めます．

〝SORT．ASM〞というファイル名で ED を起動し，インサート・モード（Ｉコマンド）に入ります．

```
A>ED SORT.ASM ↲

NEW FILE
     : *i ↲ ------小文字のｉで.
    1:
```

Figure-5.3.2
ソース・ファイルの作成．

インサート・モードに入ったら，次に示すソース・リストをキーインして行きます．

```
;==============================================================================
;               SORT PROGRAM for CP/M Learning System #2
;   This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
;======================================================== by Y.MURASE ===

        ORG     100H
DATTOP  EQU     4000H               ;DATA AREA TOP ADDRESS
DATEND  EQU     4FFFH               ;DATA AREA END ADDRESS
ENDADR  EQU     (DATEND+1) SHR 8      ;HIGH 8 BIT OF NEXT '4FFF' = '50'
BDOS    EQU     0005H               ;SYSTEM CALL ENTRY POINT

        LXI     D,MSGSTRT           ;)
        MVI     C,9                 ;)
        CALL    BDOS                ;) START MESSAGE OUT SYSTEM CALL

        LXI     D,DATTOP            ;SET TOP ADR IN D,E
        LXI     H,DATTOP            ;SET TOP ADR IN H,L
        MVI     C,0                 ;SET 0 IN C
NEXT1:
        MOV     A,M                 ;GET DATA
        CMP     C                   ;COMPARE WITH C
        JZ      SORT                ;IF A=C JUMP SORT:
;                                 *THIS STEP NOT A=C, GO NEXT DATA
        INX     H                   ;SET HL FOR NEXT DATA ADR
;                                 *CHECK OF DATA END
NEXT2:  MOV     A,H                 ;GET HIGH 8 BIT OF NEXT DATA ADR
        CPI     ENDADR              ;COMRARE WITH HIGH 8 BIT OF BUF END ADR
        JNZ     NEXT1               ;IF NOT END, GO NEXT DATA
;                                 *THIS STEP END OF DATA BUFFER
        INR     C                   ;SET NEXT PATTERN
        JZ      DONE                ;IF C=0 JUMP PROGRAM END
;                                 *THIS STEP C=NEXT PATTERN, CONTINUE
        MOV     H,D
        MOV     L,E                 ;)ADJUST D,E = H,L
        JMP     NEXT1               ;C=NEXT PATTERN AND GO LOOP

;                                 *THIS STEP A=C, SORT THE DATA
SORT:
        LDAX    D                   ;)
        MOV     M,A                 ;)
        MOV     A,C                 ;)
        STAX    D                   ;)DONE ONE DATA EXCHANGE PROCESS
        INX     D                   ;INCREMENT SORT DATA STORE POINTER
        INX     H                   ;INCREMENT DATA POINTER
        JMP     NEXT2               ;JUMP CHECK DATA END ROUTINE

;                                 *PROGRAM END, RETURN TO CP/M
DONE:
        LXI     D,MSGEND            ;)
        MVI     C,9                 ;)
        CALL    BDOS                ;) END MESSAGE OUT SYSTEM CALL

        RET                         ;END OF THIS PROGRAM. RETURN TO CP/M

MSGSTRT: DB     0DH,0AH,'SORT PROGRAM START NOW......',0DH,0AH,'$'
MSGEND:  DB     0DH,0AH,'FUNCTION COMPLETE',0DH,0AH,'$'

        END
```

• Figure-5.3.3 “SORT”のアセンブリ・ソース・プログラム

EDの機能をフルに活用し，ソース・ファイルが一応完成したら次のステップに進みます．この時点では〝SORT〟に関して，次の2つのファイルが出来ていることでしょう．

```
A>DIR SORT.* ↲
A: SORT     ASM : SORT     BAK
A>
```

Figure-5.3.4
ソース・ファイルが出来上った時点での〝SORT〟に関するファイル．

5.4 ソース・ファイルのアセンブル

アセンブルの実行をFigure-5.4.1に示します．

```
A>ASM SORT ↲
CP/M ASSEMBLER - VER 2.0
016E
000H USE FACTOR
END OF ASSEMBLY

A>
```

Figure-5.4.1
アセンブルの実行．

この例では，アセンブル・エラーは発生せず，無事にアセンブルが終了しました．出来上ったオブジェクト・コードが100Hから16DHの6EHバイト長であることなどが示されています．もしアセンブル・エラーがある場合は，そのつどエラー・メッセージが出力され，また，アセンブルにより生成されたPRNファイルを見ることによっても，リスト上のどこのラインが何のエラーであるかを知ることができます．

アセンブルが終了した時点ではFigure-5.4.2に示すように，HEXとPRNのファイルが生成されています．

```
A>DIR SORT.* ↲
A: SORT     ASM : SORT     BAK : SORT     PRN : SORT     HEX
A>
```

Figure-5.4.2　全〝SORT〟ファイルの確認．

```
A>TYPE SORT.PRN

              ;=====================================================================
              ;               SORT PROGRAM for CP/M Learning System #2
              ;    This program SORT 8 bit data at address 4000H to 4FFFH (4K bytes)
              ;===================================================== by Y.MURASE ===

0100                      ORG     100H
4000 =          DATTOP    EQU     4000H          ;DATA AREA TOP ADDRESS
4FFF =          DATEND    EQU     4FFFH          ;DATA AREA END ADDRESS
0050 =          ENDADR    EQU     (DATEND+1) SHR 8   ;HIGH 8 BIT OF NEXT '4FFF' = '50'
0005 =          BDOS      EQU     0005H          ;SYSTEM CALL ENTRY POINT

0100 113701               LXI     D,MSGSTRT      ;)
0103 0E09                 MVI     C,9            ;)
0105 CD0500               CALL    BDOS           ;) START MESSAGE OUT SYSTEM CALL

0108 110040               LXI     D,DATTOP       ;SET TOP ADR IN D,E
010B 210040               LXI     H,DATTOP       ;SET TOP ADR IN H,L
010E 0E00                 MVI     C,0            ;SET 0 IN C
                NEXT1:
0110 7E                   MOV     A,M            ;GET DATA
0111 B9                   CMP     C              ;COMPARE WITH C
0112 CA2501               JZ      SORT           ;IF A=C JUMP SORT:
                ;                               *THIS STEP NOT A=C, GO NEXT DATA
0115 23                   INX     H              ;SET HL FOR NEXT DATA ADR
                ;                               *CHECK OF DATA END
0116 7C         NEXT2:    MOV     A,H            ;GET HIGH 8 BIT OF NEXT DATA ADR
0117 FE50                 CPI     ENDADR         ;COMRARE WITH HIGH 8 BIT OF BUF END ADR
0119 C21001               JNZ     NEXT1          ;IF NOT END, GO NEXT DATA
                ;                               *THIS STEP END OF DATA BUFFER
011C 0C                   INR     C              ;SET NEXT PATTERN
011D CA2E01               JZ      DONE           ;IF C=0 JUMP PROGRAM END
                ;                               *THIS STEP C=NEXT PATTERN, CONTINUE
0120 62                   MOV     H,D
0121 6B                   MOV     L,E            ;)ADJUST D,E = H,L
0122 C31001               JMP     NEXT1          ;C=NEXT PATTERN AND GO LOOP

                ;                               *THIS STEP A=C, SORT THE DATA
                SORT:
0125 1A                   LDAX    D              ;)
0126 77                   MOV     M,A            ;)
0127 79                   MOV     A,C            ;)
0128 12                   STAX    D              ;)DONE ONE DATA EXCHANGE PROCESS
0129 13                   INX     D              ;INCREMENT SORT DATA STORE POINTER
012A 23                   INX     H              ;INCREMENT DATA POINTER
012B C31601               JMP     NEXT2          ;JUMP CHECK DATA END ROUTINE

                ;                               *PROGRAM END, RETURN TO CP/M
                DONE:
012E 115801               LXI     D,MSGEND       ;)
0131 0E09                 MVI     C,9            ;)
0133 CD0500               CALL    BDOS           ;) END MESSAGE OUT SYSTEM CALL

0136 C9                   RET                    ;END OF THIS PROGRAM. RETURN TO CP/M

0137 0D0A534F52MSGSTRT:   DB      0DH,0AH,'SORT PROGRAM START NOW......',0DH,0AH,'$'
0158 0D0A46554EMSGEND:    DB      0DH,0AH,'FUNCTION COMPLETE',0DH,0AH,'$'

016E                      END
```

Figure-5.4.3　PRN ファイルのタイプアウト.

5.5 DDTによるデバッグ

プログラムのデバッグを行う前に，まず4000H～4FFFH間のメモリに，適当にランダムなデータを用意しておかなくてはなりません．本来ならば，このエリアには，データ・エントリ・システムなどからのデータが入力され，それをこの"SORT"プログラムが値の小さい順に整理する，ということになるわけです．

ランダムなデータとして，ここでのデバッグ用には"DDT．COM"自身のマシン語コードを代用して4000Hからロードします．

その実行とデータの確認をFigure-5.5.1に示します．

```
A>DDT ↲ ------- ランダム・データ作りのためにDDTを起動.

DDT VERS 2.2
-F3000,6000,00 ↲ ----- 3000H～6000Hを0クリア.
-IDDT.COM ↲  }
-R3F00 ↲     } IとRコマンドで，DDT.COMを4000Hからロードする.
NEXT  PC
5300 0100
-F5000,6000,00 ↲ ----- 不用になった5000H～6000Hを再度0クリア.
-D3FD0,502F ↲ ---------- 出来上ったランダム・データの確認.

3FD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3FE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
3FF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
  ┌── ここからのランダム・データがSORTプログラムでソートされる.
4000 01 BC 0F C3 3D 01 43 4F 50 59 52 49 47 48 54 20 ....=.COPYRIGHT
4010 28 43 29 20 31 39 38 30 2C 20 44 49 47 49 54 41 (C) 1980, DIGITA
4020 4C 20 52 45 53 45 41 52 43 48 20 20 20 20 20 20 L RESEARCH
4030 44 44 54 20 56 45 52 53 20 32 2E 32 24 31 00 02 DDT VERS 2.2$1..
4040 C5 C5 11 30 01 0E 09 CD 05 00 C1 21 07 00 7E 3D ...0.......!..~=
4050 90 57 1E 00 D5 21 00 02 78 B1 CA 65 01 0B 7E 12 .W...!..x..e..~.
4060 13 23 C3 58 01 D1 C1 E5 62 78 B1 CA 87 01 0B 7B .#.X....bx.....{
4070 E6 07 C2 7A 01 E3 7E 23 E3 6F 7D 17 6F D2 83 01 ...z..~#.o}.o...
4080 1A 84 12 13 C3 69 01 D1 2E 00 E9 0E 10 CD 05 00 .....i..........
4090 32 5F 1E C9 21 66 1E 70 2B 71 2A 65 1E EB 0E 11 2_..!f.p+q*e....
40A0 CD 05 00 32 5F 1E C9 11 00 00 0E 12 CD 05 00 32 ...2_..........2
40B0 5F 1E C9 21 68 1E 70 2B 71 2A 67 1E EB 0E 13 CD _..!h.p+q*g.....
40C0 05 00 C9 21 6A 1E 70 2B 71 2A 69 1E EB 0E 14 CD ...!j.p+q*i.....
40D0 05 00 C9 21 6C 1E 70 2B 71 2A 6B 1E EB 0E 15 CD ...!l.p+q*k.....
40E0 05 00 C9 21 6E 1E 70 2B 71 2A 6D 1E EB 0E 16 CD ...!n.p+q*m.....
40F0 05 00 32 5F 1E C9 21 70 1E 70 2B 71 2A 6F 1E EB ..2_..!p.p+q*o..
  .
  .
  .
  .
  .
4760 41 20 42 20 20 20 44 20 20 20 48 20 20 20 53 50 A B   D   H   SP
4770 20 20 50 53 57 20 3F 3F 3D 20 1E 4D 06 00 21 45   PSW ??= .M..!E
```

```
4780 C3 A2 06 C3 AA 06 C3 CF 0D C3 B6 0B C3 DD 0B C3 ................
4790 C7 0B C3 05 0C C3 2D 0C C3 90 0C C3 66 0C C3 1F ......-.....f...
47A0 0C C9 E3 22 4A 0F E3 C3 00 00 2A 06 00 22 A8 06 ..."J.....*.."..
47B0 21 A2 06 22 01 00 21 00 00 22 06 00 AF 32 4F 0F !.."..!.."...2O.
47C0 21 00 01 22 0C 00 22 5D 0F 22 87 0F 22 B9 0F 21 !..".."]."..".."!
47D0 00 01 31 B7 0F E5 21 02 00 E5 2B 2B 22 B7 0F E5 ..1...!...++"...
47E0 E5 22 4D 0F 3E C3 32 38 00 21 86 06 22 39 00 3A ."M.>.28.!.."9.:
47F0 5D 00 FE 20 CA FE 06 21 00 00 E5 C3 AD 09 31 AF ].. ...!......1.
4800 0F CD 93 09 DA 0D 07 21 80 06 22 06 00 CD 15 0C .......!..".....
4810 3E 2D CD C7 0B CD B6 0B CD DD 0B FE 0D CA FE 06 >-..............
4820 D6 41 DA AB 0B FE 1A D2 AB 0B 5F 16 00 21 37 07 .A........_..!7.
4830 19 19 5E 23 56 EB E9 7E 07 AB 0B AB 0B C6 07 AB ..^#V..~........
4840 0B 5C 08 70 08 DA 08 04 09 AB 0B AB 0B 97 07 5A .¥.p...........Z
4850 09 AB 0B AB 0B AB 0B AB 0B 9C 09 7A 0A C3 0A BF ...........z....
  .
  .
  .
  .
  .
4F00 78 C3 F3 0D E1 F1 CA 28 0E 23 22 B9 0F EB 21 A8 X......(.#"...!.
4F10 06 4E 23 46 CD 57 08 DA 28 0E CD C8 0D 2A 4A 0F .N#F.W..(....*J.
4F20 EB 3E 82 B7 37 C3 85 08 FB 2A 4D 0F 7C B5 CA 4E .>..7....*M.|..N
4F30 0E 2B 22 4D 0F CD 1F 0C C2 4E 0E 3A 4C 0F B7 C2 .+"M.....N.:L...
4F40 48 0E CD 85 0E C3 85 08 CD 44 0D C3 85 08 CD C8 H........D......
4F50 0D 3E 2A CD C7 0B 2A B9 0F CD 93 09 D2 62 0E 22 .>*...*......B."
4F60 0C 00 CD 2E 0C 2A B7 0F 22 5D 0F C3 FE 06 11 0D .....*.."]......
4F70 00 21 2F 0F 7E A0 23 BE 23 CA 81 0E 14 1D C2 74 .!/.~.#.#......T
4F80 0E 5A 16 00 C9 2A B9 0F 46 23 E5 CD 6E 0E 21 49 .Z...*..F#..N.!I
4F90 0F 73 21 9C 0E 19 19 5E 23 56 EB E9 B8 0E E0 0E .S!....^#V......
4FA0 B8 0E E0 0E BE 0E F2 0E 04 0F 26 0F 26 0F 23 0F ..........&.&.#.
4FB0 23 0F 19 0F 26 0F 14 0F CD CE 0E C2 29 0F CD D9 #...&.......)...
4FC0 0E C3 29 0F 3A A8 06 BB C0 3A A9 06 BA C9 C1 E1 ..).:....:......
4FD0 5E 23 56 23 E5 C5 C3 C4 0E 2A B5 0F 5E 23 56 C9 ^#V#.....*..^#V.
4FE0 CD CE 0E CA ED 0E C1 C5 3E 02 C3 2B 0F D1 D5 C3 ........>..+....
4FF0 29 0F 78 FE FF C2 FC 0E AF C3 2D 0F E6 38 5F 16 ).X.......-..8_.
  └── ソート・プログラムの対象はここまで.
5000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
5010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
5020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

-^C ----- Ctrl-Cでリブートして，CP/Mに戻る.
A>
```

Figure-5.5.1　デバッグのために，ランダム・データを作る．

ソートされるデータの準備が終ったら，"SORT．HEX" のインテル HEX 形式のオブジェクト・ファイルを，DDT を使ってメモリにロードします．Figure-5.5.2に，ロードと 2～3 の DDT コマンドの実行を示します．

```
A>DDT SORT.HEX ┘ ---- DDTを起動し, ASMにより生成されたHEXオブジェクト・ファイルをロードする.

DDT VERS 2.2
NEXT  PC
016E 0000
-D100,16F ┘--- SORTプログラムのマシン語の確認.
0100 11 37 01 0E 09 CD 05 00 11 00 40 21 00 40 0E 00 .7........@!.@..
0110 7E B9 CA 25 01 23 7C FE 50 C2 10 01 0C CA 2E 01 ~..%.#|.P.......
0120 62 6B C3 10 01 1A 77 79 12 13 23 C3 16 01 11 58 bk....wy..#....X
0130 01 0E 09 CD 05 00 C9 0D 0A 53 4F 52 54 20 50 52 .........SORT PR
0140 4F 47 52 41 4D 20 53 54 41 52 54 20 4E 4F 57 2E OGRAM START NOW.
0150 2E 2E 2E 2E 2E 0D 0A 24 0D 0A 46 55 4E 43 54 49 .......$..FUNCTI
0160 4F 4E 20 43 4F 4D 50 4C 45 54 45 0D 0A 24 0B 7B ON COMPLETE..$.{

-G100,125 ┘----- 最初のデータが見つかった時点のアドレス, 125Hにブレーク・ポイントを置いて最初から実行.
SORT PROGRAM START NOW......  SORTプログラムからの出力.
*0125 ----125Hでブレークがかかった.
-T20 ┘ ---続けて20ステップをトレース実行.
C0Z1M0E1I1 A=00 B=0000 D=4000 H=403E S=0100 P=0125 LDAX D
C0Z1M0E1I1 A=01 B=0000 D=4000 H=403E S=0100 P=0126 MOV  M,A
C0Z1M0E1I1 A=01 B=0000 D=4000 H=403E S=0100 P=0127 MOV  A,C
C0Z1M0E1I1 A=00 B=0000 D=4000 H=403E S=0100 P=0128 STAX D
C0Z1M0E1I1 A=00 B=0000 D=4000 H=403E S=0100 P=0129 INX  D
C0Z1M0E1I1 A=00 B=0000 D=4001 H=403E S=0100 P=012A INX  H
C0Z1M0E1I1 A=00 B=0000 D=4001 H=403F S=0100 P=012B JMP  0116
C0Z1M0E1I1 A=00 B=0000 D=4001 H=403F S=0100 P=0116 MOV  A,H
C0Z1M0E1I1 A=40 B=0000 D=4001 H=403F S=0100 P=0117 CPI  50
C1Z0M1E1I1 A=40 B=0000 D=4001 H=403F S=0100 P=0119 JNZ  0110
C1Z0M1E1I1 A=40 B=0000 D=4001 H=403F S=0100 P=0110 MOV  A,M
C1Z0M1E1I1 A=02 B=0000 D=4001 H=403F S=0100 P=0111 CMP  C
C0Z0M0E0I1 A=02 B=0000 D=4001 H=403F S=0100 P=0112 JZ   0125
C0Z0M0E0I1 A=02 B=0000 D=4001 H=403F S=0100 P=0115 INX  H
C0Z0M0E0I1 A=02 B=0000 D=4001 H=4040 S=0100 P=0116 MOV  A,H
C0Z0M0E0I1 A=40 B=0000 D=4001 H=4040 S=0100 P=0117 CPI  50
C1Z0M1E1I1 A=40 B=0000 D=4001 H=4040 S=0100 P=0119 JNZ  0110
C1Z0M1E1I1 A=40 B=0000 D=4001 H=4040 S=0100 P=0110 MOV  A,M
C1Z0M1E1I1 A=C5 B=0000 D=4001 H=4040 S=0100 P=0111 CMP  C
C0Z0M1E1I1 A=C5 B=0000 D=4001 H=4040 S=0100 P=0112 JZ   0125
C0Z0M1E1I1 A=C5 B=0000 D=4001 H=4040 S=0100 P=0115 INX  H
C0Z0M1E1I1 A=C5 B=0000 D=4001 H=4041 S=0100 P=0116 MOV  A,H
C0Z0M1E1I1 A=40 B=0000 D=4001 H=4041 S=0100 P=0117 CPI  50
C1Z0M1E1I1 A=40 B=0000 D=4001 H=4041 S=0100 P=0119 JNZ  0110
C1Z0M1E1I1 A=40 B=0000 D=4001 H=4041 S=0100 P=0110 MOV  A,M
C1Z0M1E1I1 A=C5 B=0000 D=4001 H=4041 S=0100 P=0111 CMP  C
C0Z0M1E1I1 A=C5 B=0000 D=4001 H=4041 S=0100 P=0112 JZ   0125
C0Z0M1E1I1 A=C5 B=0000 D=4001 H=4041 S=0100 P=0115 INX  H
C0Z0M1E1I1 A=C5 B=0000 D=4001 H=4042 S=0100 P=0116 MOV  A,H
C0Z0M1E1I1 A=40 B=0000 D=4001 H=4042 S=0100 P=0117 CPI  50
C1Z0M1E1I1 A=40 B=0000 D=4001 H=4042 S=0100 P=0119 JNZ  0110
C1Z0M1E1I1 A=40 B=0000 D=4001 H=4042 S=0100 P=0110 MOV  A,M*0111

-D4000,400F ┘ ---- 上のトレースから, □のデータが入れ替ったことが分かる.
4000 [00] BC 0F C3 3D 01 43 4F 50 59 52 49 47 48 54 20 ....=.COPYRIGHT
-D4030,403F ┘
4030 44 44 54 20 56 45 52 53 20 32 2E 32 24 31 [01] 02 DDT VERS 2.2$1..
-
```

Figure-5.5.2　DDT による "SORT . HEX" のロードとデバッグ作業.

この後さらに，DDT の機能をフルに活用してデバッグ作業を行います．通常の場合は「バグあり→ED でソース・リスト修正→再アセンブル→DDT」このくり返しになります．

DDT で本プログラムをデバッグする場合，次の点に注意して下さい．本プログラムの最終出口は，CP／M にもどるために，〝RET〞（C9H）を用いています．DDT 上で，プログラムを最後まで走らせる時には，この〝RET〞のアドレスでブレークをかけるか，または〝RET〞を〝RST 7〞（FFH）に変更してから行って下さい．さもないと暴走することになります．

プログラムの修正をくり返し，バグはないと判断できれば，次に〝COM〞ファイルを作り，さらにいろいろな条件でテストを行います．（もちろん最初から COM ファイルを作って，デバッグに利用してもかまいません）．

注）　4 章の4.6「DDT コマンド」の項でさらに詳しく本プログラムのデバッグを行っています．必ず参照して下さい．

5.6　COMファイルを作り，総合テスト

プログラムが一応機能するらしいので，LOAD コマンドで，この場合の最終プログラム形体である〝SORT．COM〞を作り，総合的なテストを行います．

アセンブラにより生成された〝SORT．HEX〞から，〝SORT．COM〞に変換する LOAD コマンドの実行を，Figure-5.6.1に示します．

```
A>LOAD SORT ↲

FIRST ADDRESS 0100
LAST  ADDRESS 016D
BYTES READ    006E
RECORDS WRITTEN 01


A>
```

Figure-5.6.1
LOAD コマンドで COM ファイルを作る．

LOAD コマンドの実行が終り，メッセージに示されているように，プログラム・バイト長 6 E（HEX）のオブジェクトが出来ました．

ここで，すべての〝SORT〞に関するファイルを STAT コマンドで見てみましょう．

```
A>STAT SORT.* ↲

 Recs  Bytes  Ext Acc
   14     2k    1 R/W A:SORT.ASM
   14     2k    1 R/W A:SORT.BAK
    1     1k    1 R/W A:SORT.COM
    3     1k    1 R/W A:SORT.HEX
   22     3k    1 R/W A:SORT.PRN
Bytes Remaining On A: 204k

A>
```

Figure-5.6.2
すべての〝SORT〟ファイルの確認.

このように最終的な〝SORT.COM〟を得るために，これだけのファイルが開発途上で作られたわけです.

では，最終的な形体の〝SORT.COM〟のテストを始めましょう. その前に，再びDDTを使ってFigure-5.5.1と同様に，あらかじめランダム・データを用意しておきます.

データの用意ができたら，〝SORT.COM〟を実行します. それをFigure-5.6.3に示します.

```
A>SORT ↲

SORT PROGRAM START NOW......
この間にソート作業が行われている.
FUNCTION COMPLETE

A>
```

Figure-5.6.3
一応完成した〝SORT〟プログラムの実行.

一応，目的通りSTARTメッセージが出力されてから，少し時間がかかり，終了メッセージが出力されてCP/Mにもどっています.

問題の4000H～4FFFH間のデータはどうなったでしょう. ちゃんとソートされたでしょうか？ 実行後のデータ・エリアをDDTのダンプ・コマンドで確認したものをFigure-5.6.4に示します.

```
4000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
4090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
40A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
40B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 ................
40C0 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ................
40D0 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ................
40E0 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ................
40F0 01 01 01 01 01 01 01 01 02 02 02 02 02 02 02 02 ................
  .
  .
  .
  .
  .
4300 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B ................
4310 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0B 0C 0C 0C 0C ................
4320 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ................
4330 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ................
4340 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ................
4350 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ................
4360 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ................
4370 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ................
4380 0C 0C 0C 0C 0C 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D ................
4390 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D ................
43A0 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D ................
43B0 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D 0D ................
43C0 0D 0D 0D 0D 0D 0D 0D 0D 0D 0E 0E 0E 0E 0E 0E 0E ................
43D0 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E ................
43E0 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E ................
43F0 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E 0E ................
  .
  .
  .
  .
  .
4700 32 32 32 32 32 32 34 34 34 34 35 35 36 36 36 36 2222224444556666
4710 36 36 36 36 36 36 37 37 37 37 37 38 38 38 38 38 6666667777788888
4720 38 38 39 39 39 39 39 39 39 39 39 3A 3A 3A 3A 3A 88999999999:::::
4730 3A 3A 3A 3A 3A 3A 3B 3C 3C 3C 3C 3D 3D 3D 3D 3D ::::::;<<<<=====
4740 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D 3D ================
4750 3D 3D 3D 3E 3E 3E 3E 3E 3E 3E 3E 3E 3E 3E 3E 3E ===>>>>>>>>>>>>>
4760 3E 3E 3E 3E 3E 3E 3F 3F 3F 3F 3F 3F 40 40 40 40 >>>>>>??????@@@@
4770 40 40 40 40 40 41 41 41 41 41 41 41 41 41 41 41 @@@@@AAAAAAAAAAA
4780 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
4790 41 41 41 42 42 42 42 42 42 42 42 43 43 43 43 43 AAABBBBBBBBCCCCC
47A0 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 CCCCCCCCCCCCCCCC
47B0 43 43 43 43 44 44 44 44 44 44 44 44 44 44 44 44 CCCCDDDDDDDDDDDD
47C0 44 44 44 44 44 44 44 44 44 44 44 44 45 45 45 45 DDDDDDDDDDDDEEEE
47D0 45 45 45 45 45 45 45 45 45 45 45 45 45 45 46 46 EEEEEEEEEEEEEEFF
47E0 46 46 46 46 47 47 47 47 47 47 47 47 47 48 48 48 FFFFGGGGGGGGGHHH
47F0 48 48 48 48 48 48 48 48 48 48 48 48 49 49 49 49 HHHHHHHHHHHHIIII
  .
  .
  .
  .
  .
4B00 B4 B4 B4 B4 B4 B5 B5 B5 B6 B6 B6 B6 B7 B7 B7 B7 ................
4B10 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 ................
4B20 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 B7 ................
4B30 B7 B7 B8 B8 B9 B9 B9 B9 B9 B9 B9 B9 B9 B9 B9 B9 ................
4B40 BA BA BB BB BB BB BB BB BC BD BD BE BE BE BE BE ................
```

```
4B50 BE BE BE BE BE BE BE BE BE BE BE BF BF C0 C0 C0 ................
4B60 C0 C0 C0 C0 C0 C0 C0 C1 C1 C1 C1 C1 C1 C1 C1 C1 ................
4B70 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C2 C2 C2 C2 ................
4B80 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 ................
4B90 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 ................
4BA0 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 ................
4BB0 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 ................
4BC0 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C3 C3 C3 C3 C3 ................
4BD0 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 ................
4BE0 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 ................
4BF0 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 ................
  .
  .
  .
  .
  .
4F00 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 ................
4F10 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E6 E7 E9 E9 E9 E9 ................
4F20 E9 EA EA EB EB EB EB EB EB EB EB EB EB EB EB EB ................
4F30 EB EB EB EB EB EB EB EB EB EB EB EB EB EB EB EB ................
4F40 ED EE EE F0 F0 F0 F0 F1 F1 F1 F1 F1 F1 F1 F1 F1 ................
4F50 F1 F1 F1 F1 F1 F2 F3 F3 F3 F3 F3 F3 F3 F3 F3 F3 ................
4F60 F3 F3 F3 F3 F3 F3 F3 F3 F3 F4 F4 F4 F4 F5 F5 F5 ................
4F70 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F5 F6 F6 F6 ................
4F80 F6 F6 F6 F6 F6 F6 F6 F6 F6 F8 F9 F9 F9 F9 F9 F9 ................
4F90 F9 FA FB FB FB FB FC FC FD FE FE FE FE FE FE FE ................
4FA0 FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE ................
4FB0 FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE ................
4FC0 FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE ................
4FD0 FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE ................
4FE0 FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE FE ................
4FF0 FE FE FE FE FE FE FF FF FF FF FF FF FF FF FF FF ................

5000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
5010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
5020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Figure-5.6.4　実行後のデータの確認.

このように，きれいに小さい順に並んでいます．一見，正常に機能をしているようですが，この結果はほんとうに正しいのでしょうか．これを確かめる方法は 4000H～4FFFH 間のもとのデータを，00がいくつあるか，01はいくつあるか，02はいくつあるか，……と調べて行き，結果と比較すればよいわけです．しかし，4Kバイトものデータを人間が行うのはだいぶシンドイ話です．幸い筆者は，昔作った〝MMON〃（Mモニタ）なるものに，バイト・サーチの機能があるので，このプログラムを利用して結果のチェックを行ってみました．

再びDDTを使って，元のランダム・データを作り，〝MMON〃でチェックを行っている様子をFigure-5.6.5に示します．

```
A>MMON ⏎

-S 4000 4FFF 34 ⏎ -----サーチ・コマンドで，4000H〜4FFFH間にある "34H" のデータを捜し出す．
44EB 34 04 ┐
49B8 34 B7 │
4C8F 34 B7 │ これだけあった．
4D8D 34 E1 ┘

-S 4000 4FFF BA ⏎ -----次は "BAH".
49C7 BA C2 ┐
4FCC BA C9 ┘ これだけあった．

-S 4000 4FFF F0 ⏎ ----- 次は "F0H".
43D7 F0 0F ┐
48E3 F0 07 │
48EA F0 6F │ これだけあった．
4B34 F0 F5 ┘

-S 4000 4FFF FC ⏎ ----- 次は "FCH".
4EC0 FC FA ┐
4FF6 FC 0E ┘ これだけあった．
-
```

Figure-5.6.5　結果の詳細な検証.

このように Search 4000H〜4FFFH で，〝34″ が4個，〝BA″ が2個，〝F0″ が4個，〝FC″ が2個存在していることが，判明しました．Figure-5.6.4の結果と照し合わせると，合致することが確認されます．

プログラムは完全に機能したようです．しかしまだ安心してはいけません．「ほかのデータならどうか？」，「並び方が特別な場合は？」，などなど，あらゆるケースの場合を想定して，フローチャートの見直しや実動テストをくり返してから，やっと「実用可能」ということになるのです．

以上が，CP／M のみを使った，マシン語のソフトウェア開発の一例です．最後にこのプログラムのマシン語である〝SORT．COM″ をDUMP コマンドで示しておきます．

```
A>DUMP SORT.COM ⏎

0000 11 37 01 0E 09 CD 05 00 11 00 40 21 00 40 0E 00
0010 7E B9 CA 25 01 23 7C FE 50 C2 10 01 0C CA 2E 01
0020 62 6B C3 10 01 1A 77 79 12 13 23 C3 16 01 11 58
0030 01 0E 09 CD 05 00 C9 0D 0A 53 4F 52 54 20 50 52
0040 4F 47 52 41 4D 20 53 54 41 52 54 20 4E 4F 57 2E
0050 2E 2E 2E 2E 2E 0D 0A 24 0D 0A 46 55 4E 43 54 49
0060 4F 4E 20 43 4F 4D 50 4C 45 54 45 0D 0A 24 00 00
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

A>
```

Figure-5.6.6　ソート・プログラム〝SORT．COM″ をダンプしたもの.

注）　この〝SORT〞プログラムは，簡素化のためにスタック・ポインタをユーザー・プログラムであらたに設定せずに，CP／Mの使用しているスタック・ポインタをそのまま使っています．CCPからユーザー・プログラムに制御が移った時点では，CCP内の7レベルのスタックが使用可能です．もしこれよりスタックが深くなるプログラムでは，この方法をとることができませんので，ユーザー・プログラム内でスタック・ポインタの処理をするようにして下さい．但し，システム・コール内ではスタックを消費しません（システム・コール内部で処理される）．従ってユーザーは，システム・コール内のスタックについてまで考慮する必要はありません．

以上の開発には，CP／Mに付属の開発ツールのみを使用しましたが，マシン語開発ツールには，さらに強力なものが各社から発売されています．

例えば，デジタルリサーチ社の〝MAC〞，〝RMAC〞は，CP／M上で走らせるソフトウェア開発のための，非常に便利な各種ルーチンを，ライブラリ形式で含んだ，8080，Z80両用のマクロアセンブラであり，〝RMAC〞はさらにリロケータブル・マクロアセンブラになっています．

また同社からは，Z80用のデバッガとして，DDTとコマンド・コンパチブルなシンボリック・インストラクション・デバッガ〝ZSID〞があります（第4章 Figure-4.6.7参照）．

マイクロソフト社の〝MACRO-80〞も，8080，Z80両用のリロケータブル・マクロアセンブラであり，このアセンブラから出力されるオブジェクト形式が，8ビット・マイクロコンピュータのリロケータブル・オブジェクト形式の標準となっています．

（これらの開発ツールの使用実例は，次巻の「応用CP／M」で紹介します．）

開発ツールはその他にも，異種CPUや，16ビットCPUとのクロスアセンブラを始め，各種豊富にそろっており，この環境の良さは，CP／M以外には望めないでしょう．

あとがき

本書「実習 CP／M」により，今まで使ったこともなかった，幾つかの有用なコマンドを発見された方も多いと思います．本書は，CP／M の実用書として，内容，読みやすさ共， 読者側に立って実際に〝使える〟書として構成しました．きっと多くのユーザーから愛用していただけると思います．

本シリーズを通して，何かお気付きの点がございましたら，アスキー出版の出版部までお知らせ下されば幸いです．

CP／M シリーズ第3巻（最終編）「応用 CP／M」では，CP／M 上で走るマシン語開発に不可欠の「システム・コール　全解説」を始め，COBOL, FORTRAN, BASIC, PL／I, PL／M, PASCAL, FORTH, C, ……など，代表的な言語によるソフト開発を，同一テーマを対象に紹介する「各種高級言語によるソフトウェア開発実習」や，今話題のビジネス・プログラムの実行例紹介，それに，リロケータブル・マクロアセンブラによるモジュール別マシン語開発法，マクロライブラリの利用法などを中心に，CP／M の高度な応用について解説します．

今，CP／M を取り巻く状勢は，ますます活発になり，大手企業が本格的に CP／M への参入を開始しました．CP／M 人口はさらに急速度で拡大して行くことになるでしょう．

索　引

ラ

著者落書き

村瀬　康治

筆者にとって，アンケートはがきの批評は，よかれあしかれ，たいへんうれしいものです.「入門CP/M」について，その中の愉快な一つを紹介したいと思います.

[全文]「初めてCP/Mについて知った私にも，よくわかりました.(マイコン歴1年)もう一息くわしく，と思いますが，続巻に期待して目をつぶることにします．FM-8買ったら，CP/M走らせるからね！　大学に入るまで待ってて.」

岡山のY．Tさん，どうもありがとう．出版部のスタッフも一同，大笑いしました．しかし世間はとっても冷たく，アスキーの人達もみんな根はいい人ばかりなんですが，きっと待っていてくれないと思います．でも大丈夫．その気力さえあれば，すぐに追い付き，追い越して行けるでしょう．がんばってね.

話は変って，筆者のメイン・マシンのCP/Mは，UNIX like(？)のCP/Mなのです．このCP/Mシリーズを書くために，スクリーンOUTを必要に応じて，そのままファイルにしてしまう機能を，BIOSに追加したのです.

コンソールも〝ファイル〟になり得るUNIXに，そこだけは似ています．起動時には，〝Book CP/M〟などと，変なメッセージが出ます.

テレビ朝日技術局勤務
初版時　35才